

classes

September 23, 2020
9:53

Contents

1	Introduction	1
2	Class Attribute Descriptions	3
3	Class definitions	3
4	Element definitions	3
5	Element class attribute descriptions	4
6	Species definitions	5
7	Species class attribute descriptions	6
8	Particle definitions	7
9	Particle class attribute descriptions	9
10	Location definitions	11
11	Location class attribute descriptions	12
12	Flight definitions	13
13	Flight class attribute descriptions	15
14	Reaction definitions	16
15	Reaction class attribute descriptions	17
16	Materials definitions	18
17	Materials class attribute descriptions	19
18	Plasma-Material Interaction (PMI) definitions	21
19	PMI class attribute descriptions	22
20	Background definitions	23
21	Background class attribute descriptions	25

classes	Table of Contents	ii
22 Source definitions	26	
23 Source class attribute descriptions	31	
24 Zone definitions	38	
25 Zone class attribute descriptions	39	
26 Sector definitions	40	
27 Sector class attribute descriptions	44	
28 Detector definitions	50	
29 Detector class attribute descriptions	51	
30 Tally definitions	53	
31 Tally class attribute descriptions	58	
32 Output definitions	62	
33 Output class attribute descriptions	64	
34 Local arrays and machinery to allow compression of scoring data	66	
35 Description of compression algorithm and data structures	68	
36 Random definitions	70	
37 Cross section definitions	71	
38 Cross section class attribute descriptions	74	
39 Rate calculation definitions	76	
40 Rate calculation class attribute descriptions	78	
41 Plasma-Material Interaction (PMI) format definitions	79	
42 PMI format class attribute descriptions	83	
43 Problem definitions	85	
44 Problem class attribute descriptions	90	
45 Reaction data definitions	94	
46 Reaction data class attribute descriptions	96	
47 Plasma-Material Interaction (PMI) data definitions	99	
48 PMI data class attribute descriptions	102	
49 String definitions	106	
50 Macros for defining and manipulating arrays	106	

classes	Table of Contents	iii
51 Description of array definition macros	121	
52 Macros used to read and write netCDF files	123	
53 Description of netCDF Macros	127	
54 Arrays needed to handle particle distribution function data	128	
55 Description of particle snapshot data structures	130	
56 File names for DEGAS 2	130	
57 Description of file name usage in DEGAS 2	132	
58 Specification for a 2-D Geometry	132	
59 Two-D geometry specification class	133	
60 Sampled origin data for flights	134	
61 Description of the flight fragment data structures	137	
62 INDEX	138	

1 Introduction

\$Id: a3bb4dace5f202806b5068ac39f39d6c0a19a34d \$

This file provides a detailed description of the classes and associated common variables in the code. It is mainly intended to be read by people working on the code directly, including the authors, but may be accessible to the casual user as well. Note that because of its extensive length, the file may not be up-to-date with respect to the code. However, there should not ever be any fundamental discrepancies.

We define the various classes here and the methods to access the components. Each class has a two-letter abbreviation which is used to prefix the accessor functions. Ideally, each class has 5 standard methods:

CL_args(inst) is how to pass a instance of a class to a subroutine. *CL_dummy(inst)* is the dummy argument for a subroutine. *CL_common* will declare common variable for this class. *CL_decl(inst)* is how to declare a member of a class in a subroutine. *CL_copy(inst1, inst2)* is how to copy one instance of a class to another instance of the same class. *CL_check(inst)* will check the consistency of the instance.

This file is intended to document the “accessor functions” (usually macros), representing common arrays as well as few-line code, most used in DEGAS 2. Many of the classes below also contain descriptions of the less frequently used arrays and variables which make up the “internal” description of the class. In a genuine object-oriented language, these would correspond to “public” and “private” methods, respectively.

The instance above is designated in one of two ways.

For some classes, the information is held in global common blocks. In this case, the instance is denoted by an integer variable which is a pointer into the common block arrays. Examples of these classes are elements, species, and reactions.

For the other classes, the instance is tagged with an arbitrary string. The various components are obtained by constructing variable names from that string.

In both cases, the instance can be an array reference and an array of instances can be constructed in a natural manner; e.g., *CL_args(inst₃)* or *CL_decl(inst_{0:5})*.

These combine in a natural way. Thus if we are dealing with flight *f*, then *fl_current(f)* is the current particle being tracked. Its species is *pt_sp(fl_current(f))*. The first component of this species is *sp_el(pt_sp(fl_current(f)), 1)*, and so on.

Because DEGAS 2 is written in a non-object-oriented language and all of this class structure is implemented in a rather ad hoc way, the above described “ideals” are not adhered to throughout the code. The most familiar offenses are:

1. For those classes in which objects are denoted by a single integer, the *CL_args* and *CL_dummy* functions (which are really null macros) are frequently not used. An effort should be made to use these macros anyway as a method of labeling the variables in an argument list.
2. Again for the classes labeled by an integer, *CL_decl* is often dropped in lieu of a normal integer declaration. In some routines, the reason is that the variable in question is the index of a do-loop, and it is desirable to have its integral nature obvious to the reader. Ideally, one would have set up “conversions” from these integers to the actual class variables.

3. Some classes, e.g., the tallies, must be completely defined at the start of the run. In this case, the *CL_copy* function is not really needed and is implemented in only the most superficial manner.
4. For similar reasons, *CL_check* may not always be as thorough as it could be.

Regarding subscript notation: There is a wide variety of subscript notation in use throughout the code. Since it may never be unified, it is necessary to alert the reader to the possible forms and their interpretation.

- All macros contain argument lists in parentheses: (x, y) . These variables may either be passed into some macro code, or may only represent an abbreviation of a common variable name. For example, *el_name*(*x*) is a macro meaning *element_name*[*x*]. It *should* be the case that the order of the indices in these macro arrays corresponds to that used in C; i.e., the rightmost index is the most rapidly varying as the length of the array is traversed in memory. To be certain, one must check the macro definition.
- Direct references to common variables have subscripts in brackets: $[x]$. Through some separate coding (not explicit macros in the class declarations as above) C-like subscripts have been set up in the code so that multiple subscripts may be written as: $[y][x]$. In this case, it is guaranteed that the order of the subscripts is reversed in translating these expressions into compilable FORTRAN. The primary reason for establishing the C-like syntax is that a compact vector notation can be implemented via some well-defined macros.
- Unfortunately, the hybrid syntax $[x, y]$ is also found throughout the code. In this case, the order of the array subscripts matches that of the underlying FORTRAN array (leftmost index is most rapidly varying); usage of this sort should be discouraged.
- Finally, one can write the subscripts in the familiar FORTRAN notation: *reaction_reagent*(*i*, *x*). This is discouraged since it may possibly cause a variable reference to be mis-identified by a reader as a macro call. Also, the first two syntax options, macro or C-like, should be used whenever possible to encourage uniformity throughout the code.
- To summarize, the following are equivalent variable references:
 1. *rc_reagent*(*x*, *i*), invokes a macro,
 2. *reaction_reagent*[*i*, *x*], is the result of processing the macro,
 3. *reaction_reagent*[*x*][*i*], is the corresponding C-like syntax,
 4. *reaction_reagent*(*i*, *x*), is what the compiler will see.

– Throughout most of the rest of this document and in the “woven” code, these are typeset as (note that FWEAVE *does not* swap the index order in the third case as it should):

 1. *rc_reagent*(*x*, *i*), invokes a macro,
 2. *reaction_reagent*_{*i*, *x*}, is the result of processing the macro,
 3. *reaction_reagent*_{*x*,*i*}, is the corresponding C-like syntax,
 4. *reaction_reagent*(*i*, *x*), is what the compiler will see.
- The macro invocations used to define arrays in the header files, *define_var_pk* and *define_varp_pk*, order the subscripts in the FORTRAN way. E.g., the definition of the array used in the previous example is:

```
define_varp_pk(rc,reaction_reagent,INT,reagent_ind,reaction_ind)
```

with *i* being associated with *reagent_ind* and *x* with *reaction_ind*.

- But, netCDF is based on C conventions and the array declaration is again turned around in there:

```
int reaction_reagent(reaction_ind, reagent_ind) ;
```

The representation of the data in the file is consistent with that:

```
reaction_reagent =
 2, 3,
 2, 5,
 5, 3,
 2, 4,
```

and so on with each line specifying the reagents for a given reaction, and one line appearing for each reaction.

For more details on array definitions and the macros used to carry them out, see the documentation in *array.hweb*.

The only instance in which the macro representation of an array cannot be used is in the invocation of memory allocation arrays. These must contain the names of the actual common variables.

Throughout the code there are a handful of large, multidimensional data objects. Frequently appearing are the arrays containing the reaction and plasma-material interaction data, e.g., *reaction_handling*, and the arrays containing the tally and output data, e.g., *output_grp*. Because these dimensions are “ragged”, direct representation as multi-dimensional FORTRAN arrays is not convenient. Instead, a large, one dimensional array is dimensioned, and pointers or base address arrays used to identify the portion of the array corresponding to a particular combination of subscript values. Specific macros are invoked at setup to compute the size of the array and to establish the base addresses. Another macro is used to mimic the multi-dimensional representation for the reader’s benefit. However, the tally and output arrays are treated differently, because of the presence of additional regular structure in them See the specific section below on the output class for more details.

2 Class Attribute Descriptions

Have moved this documentation into the header files themselves (inserted into this file by FWEB) to facilitate updating documentation with code changes.

3 Class definitions

[/Users/dstotler/degas2/src/classes.hweb]

```
$Id: 6bd02e1c07485233ec227dfa1ba43ab886b4961e $
```

4 Element definitions

[/Users/dstotler/degas2/src/element.hweb]

```
$Id: bcd4c18ddad37f9e8669f74740e98c8bc44d87ab $
```

[/Users/dstotler/degas2/src/element.hweb] Information about elements is held in arrays in common blocks. An element is identified by an integer indexing into these arrays.

```
"classes.f" 4.1 ≡
@f el_decl integer
@m el_args(x) x
@m el_dummy(x) x
@m el_decl(x) integer x
@m el_copy(x,y) y = x
@m el_check(x) (x > 0 ∧ x ≤ el_num)
@m el_name(x) element_name_x
@m el_sy(x) element_sy_x
@m el_m(x) element_m_x
@m el_z(x) element_z_x
@m el_lookup(sy) string_lookup(sy, element_sy, el_num)
```

[/Users/dstotler/degas2/src/element.hweb] Length specifications.

```
"classes.f" 4.2 ≡
@m el_sy_len 3
@m el_name_len 16
```

[/Users/dstotler/degas2/src/element.hweb] Variable definitions.

```
"classes.f" 4.3 ≡
package_init(el)
define_dimen_pk(el, element_symbol_string, el_sy_len)
define_dimen_pk(el, element_name_string, el_name_len)
define_var_pk(el, el_num, INT)
define_dimen_pk(el, element_ind, el_num)
define_varp_pk(el, element_name, CHAR, element_name_string, element_ind)
define_varp_pk(el, element_sy, CHAR, element_symbol_string, element_ind)
define_varp_pk(el, element_z, INT, element_ind)
define_varp_pk(el, element_m, FLOAT, element_ind)
define_varlocal_pk(el, element_version, CHAR, string)
package_end(el)
```

5 Element class attribute descriptions

[/Users/dstotler/degas2/src/element.hweb]

[/Users/dstotler/degas2/src/element.hweb] This defines elements. Prefix is *el*.

The identifier *element* is an integer variable.

el_name(element) Returns the long name (length *el_name_len*).

el_sy(element) Returns the symbolic name (length *el_sy_len*).

el_z(element) Returns the charge number.

el_m(element) Returns the mass number (in amu).

[/Users/dstotler/degas2/src/species.hweb] Routines using elements.

el_lookup(symbol) Returns the integer element number corresponding to the symbolic name *symbol*.

6 Species definitions

[/Users/dstotler/degas2/src/species.hweb]

\$Id: 453e44e70cf6fdb69537cceaaaf7955e15a782676 \$

[/Users/dstotler/degas2/src/species.hweb] Information about species is held in arrays in common blocks. An species is identified by an integer indexing into these arrays.

```
"classes.f" 6.1 ≡
@f sp_decl integer
@m sp_args(x) x
@m sp_dummy(x) x
@m sp_decl(x) integer x
@m sp_copy(x,y) y = x
@m sp_check(x) (x > 0 ∧ x ≤ sp_num)
@m sp_name(x) species_namex
@m sp_sy(x) species_syx
@m sp_m(x) species_mx
@m sp_z(x) species_zx
@m sp_ncomp(x) species_ncompx
@m sp_generic(x) species_genericx
@m sp_multiplicity(x) species_multiplicityx
@m sp_el(x,i) species_eli,x
@m sp_count(x,i) species_counti,x
@m sp_lookup(sy) string_lookup(sy, species_sy, sp_num)
```

[/Users/dstotler/degas2/src/species.hweb] Length specifications.

```
"classes.f" 6.2 ≡
@m sp_sy_len 8
@m sp_name_len 32
@m sp_ncomp_max 10
```

[/Users/dstotler/degas2/src/species.hweb] Variable definitions.

```
"classes.f" 6.3 ≡
package_init(sp)
define_dimen_pk(sp, species_symbol_string, sp_sy_len)
define_dimen_pk(sp, species_name_string, sp_name_len)
define_var_pk(sp, sp_num, INT)
define_dimen_pk(sp, species_ind, sp_num)
define_dimen_pk(sp, species_comp_ind, sp_ncomp_max)
define_varp_pk(sp, species_name, CHAR, species_name_string, species_ind)
define_varp_pk(sp, species_sy, CHAR, species_symbol_string, species_ind)
define_varp_pk(sp, species_z, INT, species_ind)
define_varp_pk(sp, species_m, FLOAT, species_ind)
define_varp_pk(sp, species_ncomp, INT, species_ind)
define_varp_pk(sp, species_generic, INT, species_ind)
define_varp_pk(sp, species_multiplicity, INT, species_ind)
define_varp_pk(sp, species_el, INT, species_comp_ind, species_ind)
define_varp_pk(sp, species_count, INT, species_comp_ind, species_ind)
define_varlocal_pk(sp, species_version, CHAR, string)
package_end(sp)
```

7 Species class attribute descriptions

[/Users/dstotler/degas2/src/species.hweb]

[/Users/dstotler/degas2/src/species.hweb] Define species. Prefix is *sp*.

The identifier *species* is an integer variable.

sp_name(species) Returns the long name (length *sp_name_len*).

sp_sy(species) Returns the symbolic name (length *sp_sy_len*).

sp_m(species) Returns the mass (in kg).

sp_z(species) Returns the charge number.

sp_ncomp(species) Returns the number of components in a species (maximum size *sp_ncomp_max*).

sp_generic(species) Is the integer species number of the isotopic equivalent to *species* which has been designated in the species input as the archetype or “generic” member of that family.

sp_multiplicity(species) Number of different ways in which the isotopically equivalent elements in *species* could be arranged.

sp_el(species, i) Returns the element for the *i*th component.

sp_count(species, i) Returns the count of the *i*th component.

[/Users/dstotler/degas2/src/species.hweb] Routines using species.

sp_lookup(symbol) returns the integer species number corresponding to the symbolic name *symbol*.

8 Particle definitions

[/Users/dstotler/degas2/src/particle.hweb]

\$Id: 4a157ac8f2ed9dcc033e25038e47c185a35d596c \$

[/Users/dstotler/degas2/src/particle.hweb] An particle is identified by a string tag.

```
"classes.f" 8.1 ≡
@f pt_decl integer
@f pt_decls integer

@m pt_args(x)
pt_sp(x), pt_test(x), pt_t(x), pt_w(x), lc_args(pt_loc(x)), pt_v(x)_1, pt_type(x), pt_author(x)

@m pt_dummy(x)
pt_sp(x), pt_test(x), pt_t(x), pt_w(x), lc_dummy(pt_loc(x)), pt_v(x), pt_type(x), pt_author(x)

@m pt_decl(x) sp_decl(pt_sp(x))
pr_test_decl(pt_test(x))
real pt_t(x), pt_w(x), pt_v(x)_3
lc_decl(pt_loc(x))
integer pt_type(x), pt_author(x)

@m pt_decls external particle_track
logical particle_track

@m pt_copy(x, y) pt_sp(y) = pt_sp(x)
pt_test(y) = pt_test(x)
pt_t(y) = pt_t(x)
pt_w(y) = pt_w(x)
vc_copy(pt_v(x), pt_v(y))
lc_copy(pt_loc(x), pt_loc(y))
pt_type(y) = pt_type(x)
pt_author(y) = pt_author(x)

@m pt_check(x)
(sp_check(pt_sp(x)) ∧ pt_w(x) ≥ 0 ∧ lc_check(pt_loc(x))) ∧ pr_test_check(pt_test(x)) ∧ pr_test(pt_test(x)) ≡
pt_sp(x) ∧ (pt_author(x) > 0 ∧ pt_author(x) ≤ so_type_num + pr_reaction_num + pr_pmi_num)

@m pt_sp(x) pt_sp1(x)
@m pt_sp1(x) species_##x
@m pt_test(x) pt_test1(x)
@m pt_test1(x) test_##x
@m pt_t(x) pt_t1(x)
@m pt_t1(x) time_##x
@m pt_w(x) pt_w1(x)
@m pt_w1(x) weight_##x
@m pt_loc(x) pt_loc1(x)
@m pt_loc1(x) x
@m pt_v(x) pt_v1(x)
@m pt_v1(x) velocity_##x
@m pt_type(x) pt_type1(x)
@m pt_type1(x) type_##x
@m pt_author(x) pt_author1(x)
@m pt_author1(x) author_##x
```

[/Users/dstotler/degas2/src/particle.hweb] Constant definitions of particle types.

```
"classes.f" 8.2 ≡
@m pt_geometry 0
@m pt_neutral 1
@m pt_ion 2
@m zone_exit 0
@m cell_exit
```

[/Users/dstotler/degas2/src/particle.hweb] Some macros to manipulate particle tracks.

```
"classes.f" 8.3 ≡
@m pt_track(tmax, t, x) particle_track(tmax, t, pt_args(x))
@m pt_thru_face(x) lc_thru_face(pt_loc(x))
@m pt_specular(x) call surface_specular(lc_face(pt_loc(x)), lc_x(pt_loc(x))_1, pt_v(x)_1, pt_v(x)_1)
lc_face(pt_loc(x)) = 0

@m pt_author_so(x, type) pt_author(x) = type
@m pt_author_rc(x, reac) pt_author(x) = so_type_num + reac
@m pt_author_pm(x, pmi) pt_author(x) = so_type_num + pr_reaction_num + pmi
@m pt_init(x, sy, source) pt_sp(x) = sp_lookup(sy)
pt_test(x) = pr_test_lookup(pt_sp(x))
if (sp_m(pt_sp(x)) ≡ 0) then
  pt_type(x) = pt_geometry
else if (sp_z(pt_sp(x)) ≡ 0) then
  pt_type(x) = pt_neutral
  pt_author_so(x, so_type(source))
else
  pt_type(x) = pt_ion
  pt_author_so(x, so_type(source))
end if
pt_w(x) = one

@m pt_scatter_thru(x, w)
call surface_reflect(-lc_face(pt_loc(x)), lc_x(pt_loc(x))_1, pt_v(x)_1, w, pt_v(x)_1)
lc_thru_face(pt_loc(x))
```

9 Particle class attribute descriptions

[/Users/dstotler/degas2/src/particle.hweb]

[/Users/dstotler/degas2/src/particle.hweb] Define particles. Prefix is *pt*.

The identifier *particle* is a string tag.

pt_sp(particle) is the species of the particle.

pt_test(particle) is the test particle index.

pt_t(particle) is the time for this particle.

pt_w(particle) is the statistical weight of the particle.

pt_loc(particle) is the location of the particle (see below).

pt_v(particle) is the velocity of the particle.

pt_type(particle) is the type of tracking to be done for the particle.

pt_author(particle) identifies the process (reaction, plasma-material interaction, or source) responsible for setting the article's current velocity.

[/Users/dstotler/degas2/src/particle.hweb] Routines involving particles.

pt_track(tmax, t, particle) Logical function which tracks *particle* for time *tmax*, returning \mathcal{F} if a zone boundary is encountered. *pt_thru_face(particle)* updates the location of *particle* when it traverses a cell face.

pt_specular(particle) If at a face, executes a specular reflection of *particle*.

pt_author_so(particle, type) Sets the author of *particle* to source *type*.

pt_author_rc(particle, rc) Sets the author of *particle* to reaction *rc*.

pt_author_pm(particle, pm) Sets the author of *particle* to plasma-material interaction *pm*.

pt_init(particle, sy, source) Sets properties for *particle*, of species symbol *sy*, generated by source group *source*.

pt_scatter_thru(particle, v) Reflects *particle* off surface into velocity *v* relative to surface.

10 Location definitions

[/Users/dstotler/degas2/src/location.hweb]

```
$Id: 35678ba30de93ac1cad8304e0ec5e47791c11b1d $
```

A location is identified by a string tag.

```
"classes.f" 10 ≡
@f lc_decl integer
@f lc_decls integer

@m lc_args(x) lc_x(x)_1, lc_cell(x), lc_zone(x), lc_face(x), lc_cell_next(x), lc_zone_next(x),
lc_sector(x), lc_sector_next(x)

@m lc_dummy(x) lc_x(x), lc_cell(x), lc_zone(x), lc_face(x), lc_cell_next(x), lc_zone_next(x),
lc_sector(x), lc_sector_next(x)

@m lc_decl(x) real lc_x(x)_3
integer lc_cell(x), lc_zone(x), lc_face(x), lc_cell_next(x), lc_zone_next(x), lc_sector(x),
lc_sector_next(x)

@m lc_decls external locate_point, check_location
integer locate_point
logical check_location

@m lc_copy(x,y) vc_copy(lc_x(x), lc_x(y))
lc_cell(y) = lc_cell(x)
lc_zone(y) = lc_zone(x)
lc_face(y) = lc_face(x)
lc_cell_next(y) = lc_cell_next(x)
lc_zone_next(y) = lc_zone_next(x)
lc_sector(y) = lc_sector(x)
lc_sector_next(y) = lc_sector_next(x)

@m lc_check(x) check_location(lc_args(x))

@m lc_x(x) lc_x1(x)
@m lc_x1(x) pos_##x
@m lc_cell(x) lc_cell1(x)
@m lc_cell1(x) cell_##x
@m lc_zone(x) lc_zone1(x)
@m lc_zone1(x) zone_##x
@m lc_face(x) lc_face1(x)
@m lc_face1(x) surface_##x
@m lc_cell_next(x) lc_cell_next1(x)
@m lc_cell_next1(x) cell_next_##x
@m lc_zone_next(x) lc_zone_next1(x)
@m lc_zone_next1(x) zone_next_##x
@m lc_sector(x) lc_sector1(x)
@m lc_sector1(x) sector_##x
@m lc_sector_next(x) lc_sector_next1(x)
@m lc_sector_next1(x) sector_next_##x
```

[/Users/dstotler/degas2/src/location.hweb] Some interface routines

```
"classes.f" 10.1 ≡
@m lc_set(x) lc_cell(x) = locate_point(lc_x(x), lc_zone(x))
lc_face(x) = 0

@m lc_set_a(x) lc_set(x)
lc_cell_next(x) = 0
lc_zone_next(x) = 0
lc_sector(x) = 0
lc_sector_next(x) = 0
```

[/Users/dstotler/degas2/src/location.hweb] This one only sets the zone number and is used to speed up post processing scores.

```
"classes.f" 10.2 ≡
@m lc_set_b(x, z) lc_zone(x) = z
lc_zone_next(x) = 0
lc_cell(x) = 0
lc_cell_next(x) = 0
lc_sector(x) = 0
lc_sector_next(x) = 0
lc_face(x) = 0

@m lc_thru_face(x) if (lc_face(x) ≠ 0) then
    lc_face(x) = 0
    lc_cell(x) = lc_cell_next(x)
    lc_zone(x) = lc_zone_next(x)
    lc_sector(x) = lc_sector_next(x)
end if
```

[/Users/dstotler/degas2/src/location.hweb] Some constants for the geometry package.

```
"classes.f" 10.3 ≡
@m geom_epsilon const(1.0, -8) // A small distance 10 nm.
@m geom_infinity const(1.0, 16)
// A large distance—about a light year. Also a long time—about a billion years.
@m geom_large const(1000.0) // A largish (non-infinite) distance 1 km.
@m epsilon_angle const(1.0, -10) // A small angle.
```

11 Location class attribute descriptions

[/Users/dstotler/degas2/src/location.hweb]

[/Users/dstotler/degas2/src/location.hweb] Define a location. Prefix is *lc*.

The identifier *location* is a string tag.

lc_x(location) Is the position.

lc_cell(location) Is the cell number.

lc_zone(location) Is the zone number.

lc_face(location) Is the face of the cell if at a face (else 0).

lc_cell_next(location) Is the next cell number.

lc_zone_next(location) Is the next zone number.

lc_sector(location) Is the present sector number if at a sector; else 0.

lc_sector_next(location) If at or just past a sector adjacent to a second sector, contains the number of the latter; else 0.

[/Users/dstotler/degas2/src/location.hweb] Routines using locations.

lc_set(location) Sets other fields of *location* given the position.

lc_thru_face(location) Updates the fields of *location* for traversal of a cell face.

12 Flight definitions

[/Users/dstotler/degas2/src/flight.hweb]

\$Id: a7b733f02d0c4a7caf4c88af1caf955c3cb916ba \$

[/Users/dstotler/degas2/src/flight.hweb] A flight is identified by a string tag.

```
"classes.f" 12.1 ≡
@f fl_decl integer
@f fl_common integer

@m fl_args(x)
  fl_number(x), fl_source(x), fl_source_kseg(x), fl_source_xseg(x), fl_source_type(x), fl_source_root_sp(x),
  pt_args(fl_origin(x)), rn_args(fl_rand(x)), pt_args(fl_stack(x)_1), fl_pointer(x)

@m fl_dummy(x)
  fl_number(x), fl_source(x), fl_source_kseg(x), fl_source_xseg(x), fl_source_type(x), fl_source_root_sp(x),
  pt_dummy(fl_origin(x)), rn_dummy(fl_rand(x)), pt_dummy(fl_stack(x)), fl_pointer(x)

@m fl_decl(x) integer fl_number(x), fl_source(x), fl_source_kseg(x), fl_source_xseg(x), fl_source_type(x),
  fl_source_root_sp(x), fl_pointer(x)
  pt_decl(fl_origin(x))
  rn_decl(fl_rand(x))
  pt_decl(fl_stack(x)fl_stack_max)

@m fl_common integer fl_temp

@m fl_copy(x,y) fl_number(y) = fl_number(x)
  fl_source(y) = fl_source(x)
  fl_source_kseg(y) = fl_source_kseg(x)
  fl_source_xseg(y) = fl_source_xseg(x)
  fl_source_type(y) = fl_source_type(x)
  fl_source_root_sp(y) = fl_source_root_sp(x)
  pt_copy(fl_origin(x), fl_origin(y))
  rn_copy(fl_rand(x), fl_rand(y))
  fl_pointer(x) = fl_pointer(y)
  do fl_temp = 1, fl_pointer(x)
    pt_copy(fl_stack(x)fl_temp, fl_stack(y)fl_temp)
  end do

@m fl_check(x) (fl_number(x) ≥ 0 ∧ (fl_source(x) > 0) ∧ (fl_source_kseg(x) ≥ 0) ∧ (fl_source_xseg(x) ≥
  0) ∧ fl_pointer(x) > 0 ∧ fl_pointer(x) ≤ fl_stack_max ∧ pt_check(fl_current(x)))

@m fl_number(x) number##x
@m fl_source(x) source##x
@m fl_source_kseg(x) source_kseg##x
@m fl_source_xseg(x) source_xseg##x
@m fl_source_type(x) source_type##x
@m fl_source_root_sp(x) source_root_sp##x
@m fl_origin(x) origin##x
@m fl_rand(x) x
@m fl_stack(x) stack##x
@m fl_pointer(x) pointer##x
@m fl_current(x) fl_stack(x)fl_pointer(x)
```

[/Users/dstotler/degas2/src/flight.hweb] Length specifications.

```
"classes.f" 12.2 ≡
@m fl_stack_max 40
```

[/Users/dstotler/degas2/src/flight.hweb] Macros to manipulate flights.

```
"classes.f" 12.3 ≡
@m fl_label(x, num, isource) fl_number(x) = num
    fl_source(x) = isource

@m fl_init(x, sy, kseg, xseg, type, rsp) pt_init(fl_origin(x), sy, fl_source(x))
    fl_pointer(x) = 1
    fl_source_kseg(x) = kseg
    fl_source_xseg(x) = xseg
    fl_source_type(x) = type
    fl_source_root_sp(x) = rsp
    pt_copy(fl_origin(x), fl_current(x))
```

13 Flight class attribute descriptions

[/Users/dstotler/degas2/src/flight.hweb]

[/Users/dstotler/degas2/src/flight.hweb] Define a flight. Prefix is *fl*.

The identifier *flight* is a string tag.

fl_number(flight) Is the flight number.

fl_source(flight) Is the number of the source which gave rise to the flight.

fl_source_kseg(flight) Is the segment number, within this particular source group, of the source which gave rise to the flight.

fl_source_xseg(flight) Is the geometry element corresponding to the segment number which gave rise to the flight; set using *source_segment_ptr*.

fl_source_type(flight) Is the type of source initiating this flight; set using *so_type*.

fl_source_root_sp(flight) Is the “root” species associated with the source initiating this flight; set using *so_root_sp*.

fl_origin(flight) Is the particle initiating the flight.

fl_rand(flight) Is the random number information for the flight.

fl_stack(flight) Is the stack of particles being processed.

fl_pointer(flight) Is the pointer to the current particle (maximum size *fl_stack_max*).

fl_current(flight) Is the current particle (same as *fl_stack(flight)fl_pointer(flight)*).

[/Users/dstotler/degas2/src/flight.hweb] Routines using flights.

fl_label(flight, num, isource) Sets *fl_number(flight)* to *num* and

fl_source(flight) To source number *isource*.

fl_init(flight, sy, kseg, xseg, type, rsp) Sets up a current flight containing a particle of species *sy*, using *fl_origin(flight)* for the other particle properties and *fl_source(flight)* as the “author”. The source segment numbers *kseg* and *xseg* are recorded as separate parameters.

14 Reaction definitions

[/Users/dstotler/degas2/src/reaction.hweb]

\$Id: 9cb89fa72c8efd949bb0d27dd682155e8a9538e8 \$

[/Users/dstotler/degas2/src/reaction.hweb] Information about reactions is held in arrays in common blocks. An reaction is identified by an integer indexing into these arrays.

```
"classes.f" 14.1 ≡
@f rc_decl integer
@m rc_generic_no 0
@m rc_generic_yes 1
@m rc_args(x) x
@m rc_dummy(x) x
@m rc_decl(x) integer x
@m rc_copy(x,y) y = x
@m rc_check(x) (x > 0 ∧ x ≤ rc_num)
@m rc_name(x) reaction_namex
@m rc_sy(x) reaction_syx
@m rc_emitter(x) reaction_emitterx
@m rc_reagent_num(x) reaction_reagent_numx
@m rc_gen(x) reaction_genericx
@m rc_product_num(x) reaction_product_numx
@m rc_reagent(x,i) reaction_reagenti,x
@m rc_product(x,i) reaction_producti,x
@m rc_filename(x) reaction_filenamex
@m rc_reaction_type(x) reaction_typex
@m rc_lookup(sy) string_lookup(sy, reaction_sy, rc_num)
```

[/Users/dstotler/degas2/src/reaction.hweb] Length specifications.

```
"classes.f" 14.2 ≡
@M rc_sy_len 24
@M rc_name_len 80
@M rc_type_len 32
@M rc_reagent_max 2
@M rc_product_max 4
```

[/Users/dstotler/degas2/src/reaction.hweb] Variable definitions.

```
"classes.f" 14.3 ≡
package_init(rc)
define_dimen_pk(rc, reaction_symbol_string, rc_sy_len)
define_dimen_pk(rc, reaction_name_string, rc_name_len)
define_dimen_pk(rc, reaction_type_string, rc_type_len)
define_var_pk(rc, rc_num, INT)
define_dimen_pk(rc, reaction_ind, rc_num)
define_dimen_pk(rc, reaction_filename_string, FILELEN)
define_dimen_pk(rc, reagent_ind, rc_reagent_max)
define_dimen_pk(rc, product_ind, rc_product_max)
define_varp_pk(rc, reaction_name, CHAR, reaction_name_string, reaction_ind)
define_varp_pk(rc, reaction_type, CHAR, reaction_type_string, reaction_ind)
define_varp_pk(rc, reaction_sy, CHAR, reaction_symbol_string, reaction_ind)
define_varp_pk(rc, reaction_emitter, INT, reaction_ind)
define_varp_pk(rc, reaction_reagent_num, INT, reaction_ind)
define_varp_pk(rc, reaction_generic, INT, reaction_ind)
define_varp_pk(rc, reaction_product_num, INT, reaction_ind)
define_varp_pk(rc, reaction_reagent, INT, reagent_ind, reaction_ind)
define_varp_pk(rc, reaction_product, INT, product_ind, reaction_ind)
define_varp_pk(rc, reaction_filename, CHAR, reaction_filename_string, reaction_ind)
define_varlocal_pk(rc, reaction_version, CHAR, string)
package_end(rc)
```

15 Reaction class attribute descriptions

[/Users/dstotler/degas2/src/reaction.hweb]

[/Users/dstotler/degas2/src/reaction.hweb] Define reactions. Prefix is *rc*.

The identifier *reaction* is an integer variable.

rc_name(reaction) Is the name of the reaction (length *rc_name_len*).

rc_sy(reaction) Is the symbolic name of the reaction (length *rc_sy_len*).

rc_emitter(reaction) Is the reagent (if negative) or product (if positive) number to be associated with a photon emitted from this reaction.

rc_reagent_num(reaction) Is the number of reagents in the reaction (maximum value *rc_reagent_max*).

rc_reagent(reaction, i) Is the *i*th species reagent in the reaction.

rc_gen(reaction) Contains the value *rc_generic_yes* if this reaction (and its data) can be applied to species isotopically equivalent to *rc_reagent(reaction, i)*; the value *rc_generic_no* indicates that the reaction is species-specific.

rc_product_num(reaction) Is the number of products (maximum value *rc_product_max*).

rc_product(reaction, i) Is the species of the *i*th product.

rc_filename(reaction) netCDF filename containing the reaction rate and handling data (class *xs*) for the reaction.

rc_reaction_type(reaction) The reaction type determines which routines will be used to determine the reaction product velocities during run time.

[/Users/dstotler/degas2/src/reaction.hweb] Routines using reactions.

rc_lookup(symbol) returns the integer reaction number corresponding to the symbolic name *symbol*.

16 Materials definitions

[/Users/dstotler/degas2/src/materials.hweb]

\$Id: 9e49b8173b8ef2b43608f4a7b19dbb2c2ce50778 \$

[/Users/dstotler/degas2/src/materials.hweb] Information about materials is held in arrays in common blocks. A material is identified by an integer indexing into these arrays.

```
"classes.f" 16.1 ≡
@f ma_decl integer
@m ma_args(x) x
@m ma_dummy(x) x
@m ma_decl(x) integer x
@m ma_copy(x,y) y = x
@m ma_check(x) (x > 0 ∧ x ≤ ma_num)
@m ma_name(x) materials_name_x
@m ma_sy(x) materials_sy_x
@m ma_lookup(sy) string_lookup(sy, materials_sy, ma_num)
```

[/Users/dstotler/degas2/src/materials.hweb] Length specifications.

```
"classes.f" 16.2 ≡
@m ma_sy_len 8
@m ma_name_len 32
```

[/Users/dstotler/degas2/src/materials.hweb] Variable definitions.

```
"classes.f" 16.3 ≡
package_init(ma)
define_dimen_pk(ma, materials_symbol_string, ma_sy_len)
define_dimen_pk(ma, materials_name_string, ma_name_len)
define_var_pk(ma, ma_num, INT)
define_dimen_pk(ma, materials_ind, ma_num)

define_varp_pk(ma, materials_name, CHAR, materials_name_string, materials_ind)
define_varp_pk(ma, materials_sy, CHAR, materials_symbol_string, materials_ind)
define_varlocal_pk(ma, materials_version, CHAR, string)
package_end(ma)
```

17 Materials class attribute descriptions

[/Users/dstotler/degas2/src/materials.hweb]

[/Users/dstotler/degas2/src/materials.hweb] Define a material. Prefix is *ma*.

The identifier *material* is an integer variable. The properties of the materials are implicitly defined through the physical behavior of the plasma-material interactions. As a result, this class is really just an arbitrary label; a given instance of the class is generated only in conjunction with one or more PMI which refer to it.

ma_name(material) Returns the long name (length *ma_name_len*).

ma_sy(material) Returns the symbolic name (length *ma_sy_len*).

[/Users/dstotler/degas2/src/materials.hweb] Internal variables in the materials class.

ma_num Number of material instances.

[/Users/dstotler/degas2/src/materials.hweb] Routines using materials.

ma_lookup(symbol) Returns the integer materials number corresponding to the symbolic name *symbol*.

18 Plasma-Material Interaction (PMI) definitions

[/Users/dstotler/degas2/src/pmi.hweb]

\$Id: 50e851d0a56a041ce8d00b02d43563386c632bf2 \$

Information about plasma-material interaction processes is held in arrays in common blocks. A plasma-material interaction process is identified by an integer indexing into these arrays.

```
"classes.f" 18 ≡
@f pm_decl integer
@m pmi_generic_no 0
@m pmi_generic_yes 1
@m pm_args(x) x
@m pm_dummy(x) x
@m pm_decl(x) integer x
@m pm_copy(x,y) y = x
@m pm_check(x) (x > 0 ∧ x ≤ pm_num)
@m pm_name(x) pmi_name_x
@m pm_sy(x) pmi_sy_x
@m pm_reagent(x) pmi_reagent_x
@m pm_materials(x) pmi_materials_x
@m pm_gen(x) pmi_generic_x
@m pm_product_num(x) pmi_product_num_x
@m pm_product(x,i) pmi_product_i,_x
@m pm_filename(x) pmi_filename_x
@m pm_pmi_type(x) pmi_type_x
@m pm_lookup(sy) string_lookup(sy, pmi_sy, pm_num)
```

[/Users/dstotler/degas2/src/pmi.hweb] Length specifications.

```
"classes.f" 18.1 ≡
@m pm_sy_len 24
@m pm_name_len 80
@m pm_type_len 32
@m pm_product_max 4
```

[/Users/dstotler/degas2/src/pmi.hweb] Variable definitions.

```
"classes.f" 18.2 ≡
  package_init(pm)
    define_dimen_pk(pm, pmi_symbol_string, pm_sy_len)
    define_dimen_pk(pm, pmi_name_string, pm_name_len)
    define_dimen_pk(pm, pmi_type_string, pm_type_len)
    define_var_pk(pm, pm_num, INT)
    define_var_pk(pm, pm_ignore, INT)
    define_dimen_pk(pm, pmi_ind, pm_num)
    define_dimen_pk(pm, pmi_filename_string, FILELEN)
    define_dimen_pk(pm, pmi_product_ind, pm_product_max)

    define_varp_pk(pm, pmi_name, CHAR, pmi_name_string, pmi_ind)
    define_varp_pk(pm, pmi_type, CHAR, pmi_type_string, pmi_ind)
    define_varp_pk(pm, pmi_sy, CHAR, pmi_symbol_string, pmi_ind)
    define_varp_pk(pm, pmi_reagent, INT, pmi_ind)
    define_varp_pk(pm, pmi_materials, INT, pmi.ind)
    define_varp_pk(pm, pmi_generic, INT, pmi.ind)

@if 0
  define_varp_pk(pm, pmi_bound_reagent, CHAR, pmi_materials_string, pmi.ind)
  define_varp_pk(pm, pmi_bound_product, CHAR, pmi_materials_string, pmi.ind)
#endif
  define_varp_pk(pm, pmi_product_num, INT, pmi.ind)
  define_varp_pk(pm, pmi_product, INT, pmi_product.ind, pmi.ind)
  define_varp_pk(pm, pmi_filename, CHAR, pmi_filename_string, pmi.ind)
  define_varlocal_pk(pm, pmi_version, CHAR, string)
  package_end(pm)
```

19 PMI class attribute descriptions

[/Users/dstotler/degas2/src/pmi.hweb]

[/Users/dstotler/degas2/src/pmi.hweb] Define plasma-material interactions. Prefix is *pm*.

The identifier *pmi* is an integer variable.

pm_name(pmi) Returns the long name (length *pm_name_len*).

pm_sy(pmi) Returns the symbolic name (length *pm_sy_len*).

pm_reagent(pmi) Returns the species index (class *sp*) of the incident projectile.

pm_materials(pmi) Returns the materials index (class *ma*) of the target material.

pm_gen(pmi) Contains the value *pmi_generic_yes* if this *pmi* (and its data) can be applied to species isotopically equivalent to *pm_reagent(pmi)*; the value *pmi_generic_no* indicates that the *pmi* is species-specific.

pm_product.num(pmi) Number of product particles.

pm_product(pmi, i) Species index (class *sp*) of the *i*th product particle, $i = 1 \rightarrow pm_product_num(pmi)$.

pm_filename(pmi) Name of netCDF file containing the data (class *pf*) for this *pmi*.

pm_pmi_type(pmi) Returns the type of plasma-material interaction (length *pm_type_len*).

[/Users/dstotler/degas2/src/pmi.hweb] Internal variables in the PMI class.

pm_num Number of PMI in the reference database.

[/Users/dstotler/degas2/src/pmi.hweb] Routines using PMI.

pm_lookup(symbol) Returns the integer PMI number corresponding to the symbolic name *symbol*.

20 Background definitions

[/Users/dstotler/degas2/src/background.hweb]

\$Id: e5bc502c335094831b59d9646556ae4c2ac2ee7d \$

[/Users/dstotler/degas2/src/background.hweb] Information about background species is held in arrays in common blocks. An background is identified by an integer indexing into these arrays.

```
"classes.f" 20.1 ≡
@f bk_decl integer

@m bk_args(x) x
@m bk_dummy(x) x
@m bk_decl(x) integer x
@m bk_copy(x,y) y = x
@m bk_check(x) (x > 0 ∧ x ≤ bk_num)
@m bk_n(x,i) background_nx, zn_pointer(i)
@m bk_temp(x,i) background_tempx, zn_pointer(i)
@m bk_v(x,i) background_vx, zn_pointer(i)
@m plasma_coords_cartesian 1 // v holds the x, y, and z components.
@m plasma_coords_cylindrical 2 // v holds the R, φ, and z components.

@m bk_mx 120 // Dimensions for plasma code arrays
@m bk_my 120
@m bk_ms 11
@m bk_xpt 2
```

[/Users/dstotler/degas2/src/background.hweb] Move velocity from external reference plane to internal position. In cases with some cylindrical symmetry and in which the background velocities are assumed to be symmetric about the Z axis, the velocities need to be explicitly transformed back and forth between the reference plane (zero toroidal angle, or $y = 0$). The *coord* value *plasma_coords_cylindrical* denotes this symmetry option (somewhat unclearly). There may still be cases with no geometry symmetry, *geometry_symmetry_none*, that need this transformation, but disallow them for now with an assert. Note that with *geometry_symmetry_cylindrical* the velocities are always rotated back to a reference plane (zero toroidal angle) where the cartesian and cylindrical velocities are identical. Hence, the *coord* parameter is used only in cases without symmetry.

```
"classes.f" 20.2 ≡
@m v_ext_to_int(x,v,v_t,sym,coord)
if ((sym ≡ geometry_symmetry_cylindrical ∨ sym ≡ geometry_symmetry_cyl_hw ∨ sym ≡
      geometry_symmetry_cyl_section) ∧ (coord ≡ plasma_coords_cylindrical) ∧ (x12 + x22 > zero))
  then
    v_t1 = (v1 * x1 - v2 * x2) / sqrt(x12 + x22)
    v_t2 = (v1 * x2 + v2 * x1) / sqrt(x12 + x22)
    v_t3 = v3
  else
    assert(¬((sym ≡ geometry_symmetry_none) ∧ (coord ≡ plasma_coords_cylindrical)))
    vc_copy(v, v_t)
  end if
```

[/Users/dstotler/degas2/src/background.hweb] Move velocity from internal position external reference plane.

```
"classes.f" 20.3 ==
@m v_int_to_ext(x, v, v_t, sym, coord)
if ((sym ≡ geometry_symmetry_cylindrical ∨ sym ≡ geometry_symmetry_cyl_hw ∨ sym ≡
      geometry_symmetry_cyl_section) ∧ (coord ≡ plasma_coords_cylindrical) ∧ (x_1^2 + x_2^2 > zero))
then
  v_t_1 = (v_1 * x_1 + v_2 * x_2) / sqrt(x_1^2 + x_2^2)
  v_t_2 = (-v_1 * x_2 + v_2 * x_1) / sqrt(x_1^2 + x_2^2)
  v_t_3 = v_3
else
  assert(¬(sym ≡ geometry_symmetry_none) ∧ (coord ≡ plasma_coords_cylindrical))
  vc_copy(v, v_t)
end if
```

[/Users/dstotler/degas2/src/background.hweb] Variable definitions.

```
"classes.f" 20.4 ==
package_init(bk)
define_var_pk(bk, bk_num, INT)
define_dimen_pk(bk, background_ind, bk_num)
define_dimen_pk(bk, bk_plasma_ind, zone_type_num_zn_plasma)
define_var_pk(bk, background_coords, INT)
define_varp_pk(bk, background_n, FLOAT, background_ind, bk_plasma_ind)
define_varp_pk(bk, background_v, FLOAT, vector, background_ind, bk_plasma_ind)
define_varp_pk(bk, background_temp, FLOAT, background_ind, bk_plasma_ind)
package_end(bk)
```

21 Background class attribute descriptions

[/Users/dstotler/degas2/src/background.hweb]

[/Users/dstotler/degas2/src/background.hweb] Store background properties. Prefix is *bk*.

The identifier *background* is an integer variable.

bk_n(*background*, *i*) Is the density in the *i*th zone.

bk_v(*background*, *i*) Is the vector velocity in the *i*th zone.

bk_temp(*background*, *i*) Is the temperature in the *i*th zone.

background_coords Specifies the coordinate system used to represent *bk_v*, either *plasma_coords_cartesian* or *plasma_coords_cylindrical*.

22 Source definitions

[/Users/dstotler/degas2/src/sources.hweb]

```
$Id: 5bb61c9b8ceb7b4bad7c2eae78eb9950a6c412d1 $
```

[/Users/dstotler/degas2/src/sources.hweb] Specification of the sources. At present only surface and volume sources are handled.

```
"classes.f" 22.1 ≡
@f so_decl integer
@m so_args(x) x
@m so_dummy(x) x
@m so_decl(x) integer x
@m so_copy(x,y) y = x
@m so_check(x) (x > 0 ∧ x ≤ so_grps)
@m so_seg_check(x) (x > 0 ∧ x ≤ so_seg_tot)
@m so_base(i) source_base_ptri
@m so_nseg(i) source_num_segmentsi
@m so_type(i) source_typei
@m so_geom(i) source_geometryi
@m so_nflights(i) source_num_flightsi
@m so_chkpt(i) source_num_checkpointsi
@m so_species(i) source_speciesi
@m so_root_sp(i) source_root_speciesi
@m so_t_varn(i) source_time_variationi
@m so_tot_curr(i) source_total_currenti
@m so_wt_norm(i) source_weight_normi
@m so_scale(i) source_scale_factori
@m so_name(i) source_namei

@m so_plate 1 // Source types
@m so_puff 2
@m so_recomb 3
@m so_vol_source 4
@m so_snapshot 5
@m so_plt_e_bins 6
@m so_type_num 6
@m so_name_len 10

@m so_point 0 // Source geometries
@m so_line 1
@m so_surface 2
@m so_volume 3

@m so_random 0 // Source sampling schemes
@m so_direct 1

@m so_delta_fn 0 // Source time variation
@m so_time_uniform 1

@m so_e_bins_spacing_unknown 0 // Spacing option for energy distributions
@m so_e_bins_spacing_linear 1
@m so_e_bins_spacing_log 2
@m so_e_bins_num_max 15 // Maximum number of energy bins (for dimensions)

@m so_gparam_unknown 0 // Labels for global source parameters
@m so_gparam_puff_temp 1
@m so_gparam_puff_exponent 2
```

```

@m so_gparam_e_bins_min 3 // e_bins: source energy distribution in bins
@m so_gparam_e_bins_delta 4

@m so_giparam_e_bins_num 1
@m so_giparam_e_bins_spacing 2

@m so_param_unknown 0 // Labels for segment source parameters
@m so_param_v1 1 // Volume source
@m so_param_v2 2
@m so_param_v3 3
@m so_param_temperature 4
@m so_param_e_ion_delta 5 // Sheath (development purposes only for now)
@m so_param_e_ion_mult 6
@m so_param_e_ion_sheath 7
@m so_param_pt_t 8 // For snapshot source
@m so_param_pt_v1 9
@m so_param_pt_v2 10
@m so_param_pt_v3 11
@m so_param_lc_x1 12
@m so_param_lc_x2 13
@m so_param_lc_x3 14
@m so_param_e_bin_prob 15 // Source energy distribution probabilities

@m so_iparam_pt_sp 1 // Other snapshot integer data
@m so_iparam_pt_test 2 // meaningless for source particles.
@m so_iparam_lc_cell 3
@m so_iparam_lc_zone 4
@m so_iparam_pt_type 5
@m so_iparam_pt_author 6

@m so_gparams_list(i,g) source_gparameters_listsource_gparameters_baseg+i
@m so_gparams_data(i,g) source_gparameters_datasource_gparameters_baseg+i

@m so_params_list(i,g) source_parameters_listsource_parameters_baseg+i /* Note that s here is the total
segment index so that so_base(g) must be subtracted by the macro. */
@m so_params_data(i,s,g)
source_parameters_datasource_parameters_data_baseg+(s-so_base(g))*source_num_parametersg+i

@m so_giparams_list(i,g) source_giparameters_listsource_giparameters_baseg+i
@m so_giparams_data(i,g) source_giparameters_datasource_giparameters_baseg+i

@m so_iparams_list(i,g) source_iparameters_listsource_iparameters_baseg+i
@m so_iparams_data(i,s,g)
source_iparameters_datasource_iparameters_data_baseg+(s-so_base(g))*source_num_iparametersg+i

@m so_direct_mult half * (sqrt(const(5.)) - one)

```

[/Users/dstotler/degas2/src/sources.hweb] Standard initialization statements. These should eventually be in a “run control” dialog along with the number of flights, etc.

```
"classes.f" 22.2 ≡
@m so_set_run_flags so_restart = FALSE
so_seed_decimal = '12'
so_spaced_seeds = FALSE
so_seed_spacing = 100000000
so_sampling = so_random
so_time_dependent = FALSE
so_time_initialization = FALSE
so_time_initial = const(0.0)
so_time_final = const(0.0)
so_name(so_plate) = 'plate'
so_name(so_puff) = 'puff'
so_name(so_recomb) = 'recomb'
so_name(so_vol_source) = 'vol_source'
so_name(so_snapshot) = 'snapshot'
so_name(so_plt_e_bins) = 'plt_e_bins'
so_rel_wt_min = const(5., -1)
so_rel_wt_max = const(2.)
so_wt_norm_min = const(5., -1)
so_wt_norm_max = const(2.)
```

[/Users/dstotler/degas2/src/sources.hweb] Definition of the common blocks.

```
"classes.f" 22.3 ≡
package_init(so)

define_var_pk(so, so_grps, INT)
define_var_pk(so, so_seg_tot, INT)
define_var_pk(so, so_restart, INT)
define_var_pk(so, so_spaced_seeds, INT)
define_var_pk(so, so_seed_spacing, INT)
define_var_pk(so, so_sampling, INT)
define_var_pk(so, so_time_dependent, INT)
define_var_pk(so, so_time_initialization, INT)
define_var_pk(so, so_time_initial, FLOAT)
define_var_pk(so, so_time_final, FLOAT)
define_var_pk(so, so_rel_wt_min, FLOAT)
define_var_pk(so, so_rel_wt_max, FLOAT)
define_var_pk(so, so_wt_norm_min, FLOAT)
define_var_pk(so, so_wt_norm_max, FLOAT)

define_var_pk(so, so_gparams_list_size, INT)
define_var_pk(so, so_gparams_list_dim, INT)
define_var_pk(so, so_params_list_size, INT)
define_var_pk(so, so_params_list_dim, INT)
define_var_pk(so, so_params_data_size, INT)
define_var_pk(so, so_params_data_dim, INT)
define_var_pk(so, so_giparams_list_size, INT)
define_var_pk(so, so_giparams_list_dim, INT)
define_var_pk(so, so_iparams_list_size, INT)
define_var_pk(so, so_iparams_list_dim, INT)
define_var_pk(so, so_iparams_data_size, INT)
define_var_pk(so, so_iparams_data_dim, INT)

define_dimen_pk(so, source_grp_ind, so_grps)
define_dimen_pk(so, source_seg_ind, so_seg_tot)
define_dimen_pk(so, so_seed_decimal_ind, ran_c)
define_dimen_pk(so, so_name_ind, so_name_len)
define_dimen_pk(so, so_type_ind, so_type_num)
define_dimen_pk(so, source_gparams_ind, so_gparams_list_dim)
define_dimen_pk(so, source_params_ind, so_params_list_dim)
define_dimen_pk(so, source_params_data_ind, so_params_data_dim)
define_dimen_pk(so, source_giparams_ind, so_giparams_list_dim)
define_dimen_pk(so, source_iparams_ind, so_iparams_list_dim)
define_dimen_pk(so, source_iparams_data_ind, so_iparams_data_dim)

define_var_pk(so, so_seed_decimal, CHAR, so_seed_decimal_ind)
define_var_pk(so, source_name, CHAR, so_name_ind, so_type_ind)

define_varp_pk(so, source_base_ptr, INT, source_grp_ind)
define_varp_pk(so, source_num_segments, INT, source_grp_ind)
define_varp_pk(so, source_type, INT, source_grp_ind)
define_varp_pk(so, source_geometry, INT, source_grp_ind)
define_varp_pk(so, source_num_flights, INT, source_grp_ind)
define_varp_pk(so, source_num_checkpoints, INT, source_grp_ind)
define_varp_pk(so, source_species, INT, source_grp_ind)
define_varp_pk(so, source_root_species, INT, source_grp_ind)
```

```

define_varp_pk(so, source_time_variation, INT, source_grp_ind)
define_varp_pk(so, source_num_gparameters, INT, source_grp_ind)
define_varp_pk(so, source_num_parameters, INT, source_grp_ind)
define_varp_pk(so, source_gparameters_list, INT, source_gparams_ind)
define_varp_pk(so, source_parameters_list, INT, source_params_ind)
define_varp_pk(so, source_gparameters_base, INT, source_grp_ind)
define_varp_pk(so, source_parameters_base, INT, source_grp_ind)
define_varp_pk(so, source_parameters_data_base, INT, source_grp_ind)
define_varp_pk(so, source_gparameters_data, FLOAT, source_gparams_ind)
define_varp_pk(so, source_parameters_data, FLOAT, source_params_data_ind)
define_varp_pk(so, source_num_giparameters, INT, source_grp_ind)
define_varp_pk(so, source_num_iparameters, INT, source_grp_ind)
define_varp_pk(so, source_giparameters_list, INT, source_giparams_ind)
define_varp_pk(so, source_iparameters_list, INT, source_iparams_ind)
define_varp_pk(so, source_giparameters_base, INT, source_grp_ind)
define_varp_pk(so, source_iparameters_base, INT, source_grp_ind)
define_varp_pk(so, source_iparameters_data_base, INT, source_grp_ind)
define_varp_pk(so, source_giparameters_data, INT, source_giparams_ind)
define_varp_pk(so, source_iparameters_data, INT, source_iparams_data_ind)

define_varp_pk(so, source_total_current, FLOAT, source_grp_ind)
define_varp_pk(so, source_weight_norm, FLOAT, source_grp_ind)
define_varp_pk(so, source_scale_factor, FLOAT, source_grp_ind)

define_varp_pk(so, source_segment_ptr, INT, source_seg_ind)
define_varp_pk(so, source_current, FLOAT, source_seg_ind)
define_varp_pk(so, source_segment_rel_wt, FLOAT, source_seg_ind)
define_varp_pk(so, source_segment_prob_alias, FLOAT, source_seg_ind)
define_varp_pk(so, source_segment_ptr_alias, INT, source_seg_ind)
define_varlocal_pk(so, so_direct_delta, FLOAT)

package_end(so)

```

23 Source class attribute descriptions

[/Users/dstotler/degas2/src/sources.hweb]

[*/Users/dstotler/degas2/src/sources.hweb*] Define properties of sources. Prefix is *so*.

Individual source elements or “segments” are sorted into source “groups”. Surface source groups may be aligned with the sector strata, if desired, for external labeling purposes. Both the source group and segment are represented by integer variables. The segment is a general concept meant to account for all possible source geometries. For example, axisymmetric surface and volume sources are presently treated. The segments in this case represent sectors and zones, respectively. All segments in a group are assumed to have the same type, geometry, and species.

Segment-specific data are stored in single 1-D arrays with the beginning of the portion corresponding to a particular group denoted by the *so_base(igroup)* array.

A number of run control type variables and switches have been collected into this file, starting with the number of flights which sensibly belongs here. These may eventually be moved elsewhere or be assigned nontrivial values as part of a run setup procedure executed just prior to the running of the main DEGAS 2 code, *flighttest*.

so_base(igroup) Pointer in arrays defined over all segments at which the segments of group *igroup* begin.

so_nseg(igroup) Number of segments in group *igroup*.

so_type(igroup) Physical process associated with group *igroup*. Presently treat plate (recycling), gas puff, recombination, volume (*vol_source*, a specified volumetric source distinct from recombination), snapshot, and plate with binned energy distribution (*plt_e_bins*, another approach to specifying a recycling source). Additional miscellaneous source parameters (with names containing *e_bins*) specify the energy distribution used in sampling the *plt_e_bins* source. A recombination source will be set up automatically by the code when a recombining reaction is present in the problem specification. Likewise, a snapshot source is set up automatically in time dependent runs when a *snapshotfile* netCDF file is present from a previous time step.

so_geom(igroup) Geometry of group *igroup*. Presently treat axisymmetric surface and volume sources.

Although in general the geometry should be independent of the source type, practically we expect plate and puff sources to have surface geometry. Both recombination and volume sources are sampled randomly throughout the volume (according to source strength). Snapshot sources are effectively volumetric, but the source locations are determined by the particles stored in and sampled from the data in the *snapshotfile*.

so_nflights(igroup) Number of flights to be used for group *igroup* during the run. These may be set independently of the source current for each group. The variance in the observable parameters of interest due to a particular group should also be used to choose the number of flights.

so_chkpt(igroup) Number of checkpoints which should be written out for group *igroup*. A value of 0 will result in no checkpoint files.

so_species(igroup) Species index (species class) of group *igroup*.

so_root_sp(igroup) Root species that gives rise to the actual species sourced. In the cases of plate and recombination (and, perhaps, puff) sources, this is a background species providing the distribution function for the source.

so_time_varn(igroup) Specifies the time variation of group *igroup* in a time dependent simulation. The currently supported values are *so_delta_fn* (all flights start at *so_time_initial*; this is the default) and *so_time_uniform* (the initial time for flight is uniformly sampled between *so_time_initial* and *so_time_final*).

so_tot_curr(igroup) Total current (particles per second) of all segments in group *igroup*.

so_wt_norm(igroup) Factor used to renormalize the total weight flown for group *igroup* to adjust for nonunity values of *source_segment_rel_wt*.

so_scale(igroup) Arbitrary scaling factor for the total current of group *igroup*. Setting this to a value $\neq 1$ has the same impact as rescaling all of the segment currents by this factor. This provides additional flexibility for combining source groups since it can be applied after the run has been completed.

[/Users/dstotler/degas2/src/sources.hweb] Internal variables in the sources class.

so_grps Number of source groups in the problem.

so_seg_tot Total number of source segments in the problem.

source_segment_ptr_{iseg} Integer pointer to the geometry element corresponding to segment *iseg*.

source_current_{iseg} Current (in particles per second) associated with segment *iseg*.

source_segment_rel_wt_{iseg} Arbitrary multiplier on the weight of flights launched from segment *iseg*. Nonunity values allow for importance sampling of the source segments. Biasing of the results is eliminated by dividing the probability for selecting *iseg* by this factor. The factor *so_wt_norm* is also required for this purpose.

source_segment_prob_alias_{iseg} “Probability alias” for segment *iseg*. In sampling from this source group, a random number is compared with entries in this array. For further explanation see the routines *sample_sources* and *set_prob_alias*. This is computed as part of the setup process from the source current distribution. An alternative, more basic sampling procedure is available for large source groups (e.g., a snapshot source). When that procedure is to be used, this array will contain the cumulative probability distribution.

source_segment_ptr_alias_{iseg} “Pointer alias” for segment *iseg*. Used in sampling from this source group. A random number is compared with *source_segment_prob_alias*. This leads either directly to an entry in *source_segment_ptr* or to an entry in this array which serves as an indirect pointer to the same set of geometry elements. For further explanation see the routines *sample_sources* and *set_prob_alias*. This is computed as part of the setup process from the source current distribution. An alternative, more basic sampling procedure is available for large source groups. When that procedure is to be used, all elements of this array will be *int_unused*.

so_name(itype) Name of each of the source types stored as a string for output purposes.

[/Users/dstotler/degas2/src/sources.hweb] Run control parameters.

The macro *so_set_run_flags* assigns all of these at once to the values noted above. The macro is presently invoked in *boxgen.web* and *readbackground.web* (twice). Again, these parameters may eventually be relocated and / or reset elsewhere.

so_restart If *FALSE*, the code will start a new run. If *TRUE*, the code will read an existing output file and add to the data contained therein.

so_seed_decimal Character variable containing the decimal representation of the random number seed.

so_spaced_seeds If *FALSE*, the random number seed will be advanced continuously between source groups (i.e., the original mode of operation for the code). If *TRUE*, the initial seed will be incremented *so_seed_spacing* for each subsequent source group.

so_seed_spacing The number of “flights” between the initial seeds for each source group in the case of *so_spaced_seeds = TRUE*. The code will check that this number is greater than the number of flights, preventing possible reuse of a portion of the random number sequence.

so_sampling Specifies the scheme used to sample the initial source segment for a flight. The usual, random, Monte Carlo approach is selected by *so_random*. The scheme selected by *so_direct* flight number to directly generate an irrational number between 0 and 1 which is then used in the usual fashion in extracting a segment number from the “probability alias” arrays.

so_time_dependent Is usually left at the default *FALSE* value, representing a steady state or time independent simulation. When *TRUE*, the value of *so_time_final* must be set to something larger than *so_time_initial*, determining the time interval (as experienced by the flights) over which the flights will be followed.

so_time_INITIALIZATION Is usually left at the default *FALSE* and has no effect if the time dependent parameters are not set. If *TRUE*, flights are tracked much as in a steady state run, but are stopped after each time interval *so_time_final - so_time_initial* so that a marker can be added to the snapshot distribution. The end result is a snapshot distribution of particles that can serve as the initial state for a fully time dependent run.

so_time_initial For a time dependent simulation, the smallest initial time (in seconds) that can be assigned to a flight. E.g., for a time independent source, the initial time will be randomly sampled between *so_time_initial* and *so_time_final*.

so_time_final For a time dependent simulation, the largest initial time (in seconds) that can be assigned to a flight. E.g., for a time independent source, the initial time will be randomly sampled between *so_time_initial* and *so_time_final*. More importantly, all flights will be stopped when their time reaches this value, provided they have not been already terminated due to ionization, absorption, etc.

so_rel_wt_min For iterations with a coupled plasma code only, places a lower bound on the ratio of the updated current in a source segment to the “old” current last used to set the source sampling (probability alias) arrays. Smaller values of the ratio will cause the source sampling arrays to be updated. The test is applied separately to each group.

so_rel_wt_max For iterations with a coupled plasma code only, places an upper bound on the ratio of the updated current in a source segment to the “old” current last used to set the source sampling (probability alias) arrays. Larger values of the ratio will cause the source sampling arrays to be updated. The test is applied separately to each group.

so_wt_norm_min For iterations with a coupled plasma code only, places a lower bound on the *so_wt_norm* for a source group (providing an essentially global check on *source_segment_rel_wt*). Smaller values of *so_wt_norm* will cause the source sampling arrays to be updated.

so_wt_norm_max For iterations with a coupled plasma code only, places an upper bound on the *so_wt_norm* for a source group (providing an essentially global check on *source_segment_rel_wt*). Larger values of *so_wt_norm* will cause the source sampling arrays to be updated.

[/Users/dstotler/degas2/src/sources.hweb] Parameters and arrays associated with miscellaneous source parameters.

The need for these parameters arises from the fact that the set of parameters required to specify a source group varies considerably with the type of source. In fact, even the number of parameters can vary dramatically, going from ~ 2 for a gas puff to > 10 for a snapshot source. The miscellaneous source parameters provide an extensible framework for specifying just about any conceivable source type.

These parameters are broken down into four categories, each distinguished by a different “root” phrase that appears in each of the constituent variables used to specify and store the parameters. Namely:

giparams or giparameters Global, integer parameters. These are “global” in that they are same for all segments of a particular source group.

gparams or gparameters Global, floating parameters. Again, these are the same for all segments of a particular source group.

iparams or iparameters Integer parameters that vary with each segment of a particular source group.

params or parameters Floating parameters that vary with each segment of a particular source group.

For each of these four there is an associated *num* variable that specifies the number of parameters of each source type that are needed for each source group; this may very well be zero. Then, there is a 2-D *list* variable that keeps track of what these parameters are. The entries in these lists are integers that are associated with specific physical or computational quantities via the macro values at the top of this file. The range of the first index of the *list* is governed by *num*; the second index is the source group. As is the case with all of the multi-dimensional variables, the *list* is actually stored as a 1-D array and accessed with a 2-D macro (defined above). The length being used for this array is specified by the *list_size* variable (a scalar). There is an independent dimension, *list_dim*, which is needed so that the arrays can be made non-trivial for output to the netCDF files even if the *list_size* is zero. The initial entry in the 1-D list array for each source group is contained in the *base* array.

The actual values of the parameters are similarly specified in 1-D arrays containing the word **data**. These are accessed via 2-D (for “global” parameters) or 3-D macros (for segment-varying parameters) in a similar manner. In fact, the indexing required for the global parameters uses same *base* array since the parameter value, like its identifier, is a scalar. For the segment-varying parameters, the corresponding quantities are the *data_base* arrays. Likewise, the length of the data arrays being used is *data_size* and the actual dimension, *data_dim*.

Here is the full list of the miscellaneous source parameter variables and applicable macro interfaces:

so_gparams_list_size Number of entries in the list of global parameters.

so_gparams_list_dim The dimension corresponding to *so_gparams_list_size*. Need a separate variable to allow the arrays to be nontrivial (for the sake of netCDF) when *so_gparams_list_size* = 0.

so_params_list_size Number of entries in the list of segment based parameters.

so_params_list_dim The dimension corresponding to *so_params_list_size*. Need a separate variable to allow the arrays to be nontrivial (for the sake of netCDF) when *so_params_list_size* = 0.

so_params_data_size Number of entries in the full set of segment based parameter data.

so_params_data_dim The dimension corresponding to *so_params_data_size*. Need a separate variable to allow the arrays to be nontrivial (for the sake of netCDF) when *so_params_data_size* = 0.

so_giparams_list_size Number of entries in the list of integer global parameters.

so_giparams_list_dim The dimension corresponding to *so_giparams_list_size*. Need a separate variable to allow the arrays to be nontrivial (for the sake of netCDF) when *so_giparams_list_size* = 0.

so_iparams_list_size Number of entries in the list of integer segment based parameters.

so_iparams_list_dim The dimension corresponding to *so_iparams_list_size*. Need a separate variable to allow the arrays to be nontrivial (for the sake of netCDF) when *so_iparams_list_size* = 0.

so_iparams_data_size Number of entries in the full set of integer segment based parameter data.

so_iparams_data_dim The dimension corresponding to *so_iparams_data_size*. Need a separate variable to allow the arrays to be nontrivial (for the sake of netCDF) when *so_iparams_data_size* = 0.

source_num_gparameters_{igroup} Number of global, miscellaneous parameters associated with source group *igroup*.

source_num_parameters_{igroup} Number of segment based, miscellaneous parameters associated with source group *igroup*.

so_gparams_list(iparam, igrp) Identity (e.g., *so_gparam_puff_temp*) of miscellaneous global parameter *iparam* for source group *igrp*.

so_gparams_data(iparam, igrp) Real value of miscellaneous global parameter *iparam* for source group *igrp*.

so_params_list(iparam, igrp) Identity (e.g., *so_param_temperature*) of miscellaneous segment based parameter *iparam* for source group *igrp*.

so_params_data(iparam, iseg, igrp) Real value of miscellaneous segment based parameter *iparam* for source group *igrp* at total segment *iseg* (i.e., $1 \leq iseg \leq so_seg_tot$).

source_giparameters_base_{igroup} Pointer to the starting point in the *source_giparameters_list* and *source_giparameters_data* arrays for source group *igroup*.

source_iparameters_base_{igroup} Pointer to the starting point in the *source_iparameters_list* for source group *igroup*.

source_iparameters_data_base_{igroup} Pointer to the starting point in the *source_iparameters_data* for source group *igroup*.

source_num_giparameters_{igroup} Number of integer, global, miscellaneous parameters associated with source group *igroup*.

source_num_iparameters_{igroup} Number of integer, segment based, miscellaneous parameters associated with source group *igroup*.

so_giparams_list(iparam, igrp) Identity of miscellaneous integer, global parameter *iparam* for source group *igrp*.

so_giparams_data(iparam, igrp) Real value of miscellaneous, integer, global parameter *iparam* for source group *igrp*.

so_iparams_list(iparam, igrp) Identity of miscellaneous, integer, segment based parameter *iparam* for source group *igrp*.

so_iparams_data(iparam, iseg, igrup) Real value of miscellaneous, integer, segment based parameter *iparam* for source group *igrup* at total segment *iseg* (i.e., $1 \leq iseg \leq so_seg_tot$).

source_giparameters_base_{igrup} Pointer to the starting point in the *source_giparameters_list* and *source_giparameters_data* arrays for source group *igrup*.

source_iparameters_base_{igrup} Pointer to the starting point in the *source_iparameters_list* for source group *igrup*.

source_iparameters_data_base_{igrup} Pointer to the starting point in the *source_iparameters_data* for source group *igrup*.

24 Zone definitions

[/Users/dstotler/degas2/src/zone.hweb]

\$Id: 44d7fe8a9648289a39c46451ed0e37bdd8733402 \$

[/Users/dstotler/degas2/src/zone.hweb] Information about zones is held in arrays in common blocks. An zone is identified by an integer indexing into these arrays.

```
"classes.f" 24.1 ≡
@f zn_decl integer
@m zn_args(x) x
@m zn_dummy(x) x
@m zn_decl(x) integer x
@m zn_decls logical check_zone
@m zn_copy(x,y) y = x
@m zn_check(x) check_zone(zn_args(x))
@m zn_type(x) zone_type_x
@m zn_index(x,i) zone_indexi,x
@m zn_pointer(x) zone_pointer_x
@m zn_volume(x) zone_volume_x
```

[/Users/dstotler/degas2/src/zone.hweb] Length specifications.

```
"classes.f" 24.2 ≡
@m zn_index_max 4
@m zi_ix 1 // Indices used with zone_index
@m zi_iz 2
@m zi_iy 3 // Much newer, hence, the odd ordering.
@m zi_ptr 4

@m zn_type_max 4
@m zn_undefined 0
@m zn_vacuum 1
@m zn_plasma 2
@m zn_solid 3
@m zn_exit 4
```

[/Users/dstotler/degas2/src/zone.hweb] Variable definitions.

```
"classes.f" 24.3 ≡
package_init(zn)
define_var_pk(zn, zn_num, INT)
define_dimen_pk(zn, zone_type_ind, zn_type_max)
define_dimen_pk(zn, zone_index_ind, zn_index_max)
define_dimen_pk(zn, zone_ind, zn_num)
define_var_pk(zn, zone_type_num, INT, zone_type_ind)
define_varp_pk(zn, zone_type, INT, zone_ind)
define_varp_pk(zn, zone_index, INT, zone_index_ind, zone_ind)
define_var_pk(zn, zone_index_min, INT, zone_index_ind)
define_var_pk(zn, zone_index_max, INT, zone_index_ind)
define_varp_pk(zn, zone_pointer, INT, zone_ind)
define_varp_pk(zn, zone_volume, FLOAT, zone_ind)
define_varp_pk(zn, zone_center, FLOAT, vector, zone_ind)
define_varp_pk(zn, zone_min, FLOAT, vector, zone_ind)
define_varp_pk(zn, zone_max, FLOAT, vector, zone_ind)
define_varlocal_pk(zn, zone_version, CHAR, string)
package_end(zn)
```

[/Users/dstotler/degas2/src/zone.hweb] Setting up zone information.

```
"classes.f" 24.4 ≡
@m zn_type_set(zone, type) zn_type(zone) = type
zone_type_numtype = zone_type_numtype + 1
zn_pointer(zone) = zone_type_numtype
```

25 Zone class attribute descriptions

[/Users/dstotler/degas2/src/zone.hweb]

[/Users/dstotler/degas2/src/zone.hweb] Define a zone. Prefix is *zn*.

The identifier *zone* is an integer variable.

zn_type(zone) Is one of *zn_vacuum*, *zn_plasma*, *zn_solid* indicating the type of zone.

zn_index(zone, l) For $l = 1, \dots, \rightarrow zn_index_max$ gives the fluid code indices of the zone. More specifically, $l = 1$ and 2 are for the 2-D fluid codes. $l = 3$ is an index for the third dimension in nearly symmetric 3-D cases. In these cases, the $l = 1$ and $l = 2$ values do not vary in the third dimension; this index allows them to be distinguished. The $l = 4$ index then serves as a pointer back to a reference zone, the one for which $zn_index(zone, zi_ptr) = zone$, for that set. This allows, say, 2-D information specified at a single value of the $l = 3$ index to be mapped to the others along the third dimension.

zn_pointer(zone) Is a pointer into arrays specific for different types of zones: plasma, solid, etc.

zn_volume(zone) Is the volume of the zone.

[/Users/dstotler/degas2/src/zone.hweb] Internal variables in the zone class.

zn_num Number of zones in the problem.

zone_center_zone Is an approximate center for the zone; accuracy depends on the geometry. This is always a Cartesian coordinate. In 2-D cases, the *y* coordinate is usually fixed at 0; in some 2-D plane symmetric cases it may have a (constant) nonzero value.

zone_min_zone Minimum value of the coordinates of the corners of the zone; the minimum is performed separately for each component so that this point is not necessarily part of the zone. In 1-D and 2-D cases, the *y* coordinate is always = 0. In planar 3-D cases, it corresponds to the Cartesian *y* value. In cylindrical 3-D cases, it contains an azimuthal angle in radians.

zone_max_zone Maximum value of the coordinates of the corners of the zone; the maximum is performed separately for each component so that this point is not necessarily part of the zone. In 1-D and 2-D cases, the *y* coordinate is always = 0. In planar 3-D cases, it corresponds to the Cartesian *y* value. In cylindrical 3-D cases, it contains an azimuthal angle in radians.

zone_index_min_l Minimum value of $zn_index(zone, l)$ for $l = 1, \dots, \rightarrow zn_index_max$.

zone_index_max_l Maximum value of $zn_index(zone, l)$ for $l = 1, \dots, \rightarrow zn_index_max$.

26 Sector definitions

[/Users/dstotler/degas2/src/sector.hweb]

\$Id: f6b48c6e8afbaaf00c82a9777560d279fb6ece5 \$

[/Users/dstotler/degas2/src/sector.hweb] A stratum is identified by a positive integer.

```
"classes.f" 26.1 ≡
@f sc_decl integer
@m sc_args(x) x
@m sc_dummy(x) x
@m sc_decl(x) integer x
@m sc_plasma_decl(x) integer x
@m sc_target_decl(x) integer x
@m sc_wall_decl(x) integer x
@m sc_copy(x,y) y = x
@m sc_check(x) (x > 0 ∧ x ≤ nsectors)
@m sc_compare(sect, zone, face, stratum, seg)
((zone ≡ sector_zonesect) ∧ (face ≡ sector_surfacesect) ∧ (stratum ≡ stratasect) ∧ (seg ≡
sector_strata_segmentsect))
@m sc_plasma_check(x) (x > 0 ∧ x ≤ sc_plasma_num)
@m sc_vacuum_check(x) (x > 0 ∧ x ≤ sc_vacuum_num)
@m sc_target_check(x) (x > 0 ∧ x ≤ sc_target_num)
@m sc_wall_check(x) (x > 0 ∧ x ≤ sc_wall_num)
@m sc_exit_check(x) (x > 0 ∧ x ≤ sc_exit_num)
@m sc_diag_check(x,g) (x > 0 ∧ x ≤ diagnostic_num_sectorsg)
// Define these to work around the [0] element.

@m diag_lookup(name) string_lookup(name, diagnostic_grp_name1, sc_diagnostic_grps)
@m plasma_lookup(sector) int_lookup(sector, plasma_sector1, sc_plasma_num)
@m target_lookup(sector) int_lookup(sector, target_sector1, sc_target_num)
@m wall_lookup(sector) int_lookup(sector, wall_sector1, sc_wall_num)

@m sc_unknown 0 // Sector types
@m sc_vacuum 1
@m sc_plasma 2
@m sc_target 3
@m sc_wall 4
@m sc_exit 5
@m sc_diagnostic(grp) 6 + grp - 1 // I guess this will work as long as new ones go in before here
@m sc_type_max 17

@m sc_diag_unknown 0 // Independent variables for diagnostic groups
@m sc_diag_energy 1
@m sc_diag_angle 2

@m sc_diag_spacing_unknown 0
@m sc_diag_spacing_linear 1
@m sc_diag_spacing_log 2

@m sc_diag_name_len 40

@m diagnostic_sector(i, grp) diagnostic_sector_tabdiagnostic_grp_basegrp + i - 1 // Used anywhere?
```

[/Users/dstotler/degas2/src/sector.hweb] Define the dimensions and variables used by the geometry routines. Note that the dimensions in this file begin at 0. However, the 0 index isn't really used. They are defined this way to accomodate netcdf's wish to say that a zero length dimension is unlimited. Namely, until sectors are in wide use, *nsectors*, *sc_vacuum_num*, etc. will = 0 for many cases. If the corresponding dimensions began at 1, the *length* of the dimension would be computed to be = 0; hence, netcdf would think we meant these dimensions be unlimited. We would then get an error for having more than one such dimension in the file. By instead starting them at 0, the dimensions have a length of at least 1.

```
"classes.f" 26.2 ≡
package.init(sc)
define_var_pk(sc, nsectors, INT)
define_dimen_pk(sc, sector_ind, 0, nsectors)
define_dimen_pk(sc, sector_neg_pos_ind, sc_neg, sc_pos)
define_dimen_pk(sc, sector_type_ind, sc_type_max)

/* These two provide external code labels for each sector, analogous to zone_index. E.g., strata
   would correspond to DEGAS walls and sector_strata_segment to wall segments. */
define_varp_pk(sc, strata, INT, sector_ind)
define_varp_pk(sc, sector_strata_segment, INT, sector_ind)

define_varp_pk(sc, sectors, INT, sector_ind)
define_varp_pk(sc, sector_zone, INT, sector_ind)
define_varp_pk(sc, sector_surface, INT, sector_ind)
define_varp_pk(sc, sector_points, FLOAT, vector, sector_neg_pos_ind, sector_ind)
define_varp_pk(sc, sector_type_pointer, INT, sector_type_ind, sector_ind)

/* Subclasses; first vacuum */
define_var_pk(sc, sc_vacuum_num, INT)
define_dimen_pk(sc, vacuum_ind, 0, sc_vacuum_num)
define_varp_pk(sc, vacuum_sector, INT, vacuum_ind) // Properties: ?

/* plasma subclass */
define_var_pk(sc, sc_plasma_num, INT)
define_dimen_pk(sc, plasma_ind, 0, sc_plasma_num)
define_varp_pk(sc, plasma_sector, INT, plasma_ind)

/* target subclass */
define_var_pk(sc, sc_target_num, INT)
define_dimen_pk(sc, target_ind, 0, sc_target_num)
define_varp_pk(sc, target_sector, INT, target_ind)
define_varp_pk(sc, target_material, INT, target_ind)
define_varp_pk(sc, target_temperature, FLOAT, target_ind)
define_varp_pk(sc, target_recyc_coef, FLOAT, target_ind)
// Properties: field-line-angle-of-incidence, for example

/* wall subclass */
define_var_pk(sc, sc_wall_num, INT)
define_dimen_pk(sc, wall_ind, 0, sc_wall_num)
define_varp_pk(sc, wall_sector, INT, wall_ind)
define_varp_pk(sc, wall_material, INT, wall_ind)
define_varp_pk(sc, wall_temperature, FLOAT, wall_ind)
define_varp_pk(sc, wall_recyc_coef, FLOAT, wall_ind)

/* exit subclass (includes pump) */
define_var_pk(sc, sc_exit_num, INT)
define_dimen_pk(sc, exit_ind, 0, sc_exit_num)
define_varp_pk(sc, exit_sector, INT, exit_ind)
```

```
@#if 0
    define_varp_pk(sc, exit_albedo, FLOAT, exit_ind)
@#endif

/* diagnostic subclass */
define_var_pk(sc, sc_diagnostic_grps, INT)

define_dimen_pk(sc, sc_diag_name_string, sc_diag_name_len)
define_dimen_pk(sc, diag_grp_ind, 0, sc_diagnostic_grps)

define_var_pk(sc, sc_diag_max_bins, INT)

define_varp_pk(sc, diagnostic_grp_name, CHAR, sc_diag_name_string, diag_grp_ind)
define_varp_pk(sc, diagnostic_num_sectors, INT, diag_grp_ind)

define_varp_pk(sc, diagnostic_var, INT, diag_grp_ind)

define_varp_pk(sc, diagnostic_tab_index, INT, diag_grp_ind)
define_varp_pk(sc, diagnostic_min, FLOAT, diag_grp_ind)
define_varp_pk(sc, diagnostic_delta, FLOAT, diag_grp_ind)
define_varp_pk(sc, diagnostic_spacing, INT, diag_grp_ind)

define_varp_pk(sc, diagnostic_grp_base, INT, diag_grp_ind)
define_var_pk(sc, sc_diag_size, INT)
define_dimen_pk(sc, sc_diag_ind, 0, sc_diag_size - 1)
define_varp_pk(sc, diagnostic_sector_tab, INT, sc_diag_ind)

package_end(sc)
```

```
[/Users/dstotler/degas2/src/sector.hweb] Indices

"classes.f" 26.3 ≡
@m sc_neg 0
@m sc_pos 1

/* Constructors (really?) for sector subclasses. Any real reason for not doing these as subroutines?
*/
@m define_sector_vacuum(sc) sc_vacuum_num = sc_vacuum_num + 1
var_realloc(vacuum_sector)
vacuum_sector_sc_vacuum_num = sc
sector_type_pointersc,sc_vacuum = sc_vacuum_num
@m define_sector_plasma(sc) sc_plasma_num = sc_plasma_num + 1
var_realloc(plasma_sector)
plasma_sector_sc_plasma_num = sc
sector_type_pointersc,sc_plasma = sc_plasma_num
@m define_sector_target(sc, mat, temp, recyc) sc_target_num = sc_target_num + 1
var_realloc(target_sector)
var_realloc(target_material)
var_realloc(target_temperature)
var_realloc(target_recyc_coef)
target_sector_sc_target_num = sc
sector_type_pointersc,sc_target = sc_target_num
target_material_sc_target_num = mat
target_temperature_sc_target_num = temp
target_recyc_coefsc_target_num = recyc
@m define_sector_wall(sc, mat, temp, recyc) sc_wall_num = sc_wall_num + 1
var_realloc(wall_sector)
var_realloc(wall_material)
var_realloc(wall_temperature)
var_realloc(wall_recyc_coef)
wall_sector_sc_wall_num = sc
sector_type_pointersc,sc_wall = sc_wall_num
wall_material_sc_wall_num = mat
wall_temperature_sc_wall_num = temp
wall_recyc_coefsc_wall_num = recyc
@m define_sector_exit(sc) sc_exit_num = sc_exit_num + 1
var_realloc(exit_sector)
exit_sector_sc_exit_num = sc
sector_type_pointersc,sc_exit = sc_exit_num
@if 0 exit_albedosc_exit_num = alb
#endif
```

27 Sector class attribute descriptions

[/Users/dstotler/degas2/src/sector.hweb]

[/Users/dstotler/degas2/src/sector.hweb] Define a sector. Prefix is *sc*.

The default behavior for a surface in DEGAS 2's tracking algorithm is to stop the flight momentarily (to check for a zone boundary) and then to continue. In some instances, these surfaces need to do more, e.g.,

1. If that surface represents the interface between a plasma or vacuum zone and a material (solid) or exit zone, the quantities describing the flight must be altered to represent the effect of that next zone,
2. If the user wishes to monitor the passing of a flight through a particular surface, the flight's parameters must be noted and used to update the appropriate tallies.

To address these needs, DEGAS 2 utilizes the *sector* concept. A sector is identified by a positive integer *sector*. At a minimum, a sector represents a coupled surface and a zone. More specifically, the surface should be a bounding surface of a cell comprising the zone (the orientation is important; see below). The function used to define sectors takes as an optional argument a second zone. If provided, the function will verify that the input surface is indeed an interface between the two zones. The programmer should use this option whenever possible as a consistency check.

A set of sector subclasses have been established to provide clearly defined arrays for storing sector-related properties. The intention is that the number of these subclasses will change infrequently; with new properties added occasionally. Hence, we have opted, for clarity, to make the subclass and property names explicit rather than store all of this information in some flexible array structure.

The sector subclasses are:

target The interface between a plasma zone and a solid zone; the face used is a surface of the solid zone,

wall The interface between a vacuum zone and a solid zone; the face used is a surface of the solid zone,

exit The interface between a vacuum or plasma zone and an exit zone; the face used is a surface of the exit zone ,

diagnostic A diagnostic sector.

plasma The complementary interface to a target sector, i.e., using the surface of the plasma zone.

vacuum The complementary interface to a wall sector, i.e., using the surface of the vacuum zone.

At one point, the elimination of the *plasma* and *vacuum* types was considered; the desired objectives were, at that time, being met with just a single sector at each interface. However, initial work with nearly symmetric 3-D problems made clear the need for maintaining all four types. In these cases, the zone type can vary in the third dimension so that the nature of the interface between adjacent zones can also vary. By having sectors defined on both sides, the interface can be completely characterized.

As an example, consider a vertical pipe inserted into the bottom of an otherwise toroidally symmetric solid annular plate. Particles impinging on the plate from above would strike a solid surface except at the location of the pipe; there, they would travel down into the pipe. The surface that serves as the interface between the upper volume and the surface / pipe entrance would be identical, say, $z = \text{constant}$. However, we can correctly track the particle through that interface only if we know about the 3-D structure of the plate / pipe combination. This information would be contained in the zones used to model the plate and pipe, and not the surface between them and the volume above. We can accomplish this just by defining separate sectors for the plate (say, a *wall* sector) and the pipe interior (say, a *vacuum* sector). Furthermore, the gas properties in the volume above may be varying toroidally, leading to a toroidal discretization of that vacuum region. Each of those zones could be associated with a different *vacuum* sector. These sectors could also be used to tally toroidally resolved diagnostics at the vacuum - plate / pipe interface or to define toroidally discretized sources at the plate surface. In a toroidally symmetric (2-D) problem, there is no pipe, and a single sector sufficiently defines the interface. Hence, the earlier approach of using just *wall* and *target* sectors.

The *exit* and *diagnostic* subclasses have not been qualitatively altered since their introduction. However, the never-implemented *albedo* property of the exits has been replaced with the equivalent functionality of the *wall* and *target* recycling coefficients. The *exit* sector (and the associated *zone* type) has been retained; note that particles striking an *exit* sector are killed without undergoing any PMI processing. Another useful feature of *exit* sectors is that a separate default diagnostic group is defined for them.

A *target* sector is assumed to exist at the interface between adjacent plasma and solid zones. The properties *target_material*, *target_temperature*, and *target_recyc_coef* describe the solid. Obviously, the *plasma* sector is defined using the plasma zone, and the *target* sector is defined using the target zone. The sectors' surfaces are of opposite sign, chosen so that a particle leaving the plasma zone goes "out" (in the sense determined by the surface function's gradient) through the *plasma* sector's surface and goes "into" the *target* sector's surface. (Although this also seems obvious, the previous implementation used the opposite convention.)

The *wall* and *vacuum* sectors exist at the interface between vacuum and solid zones. They currently differ from the *target* and *plasma* zones only by the absence of the above-mentioned sheath related properties.

The diagnostic subclass (distinct from the separate, but also arbitrarily named, *detector* class) properties permit sectors to record information about sector-crossing flights. Default diagnostic sectors are set up using the declared *target*, *wall*, and *exit* sectors. Associated tallies are defined from these. Related diagnostic sectors with identical properties may be collectively "grouped" together. There may be several groups of diagnostics in a given problem: one to measure particle current to each sector, one to collect an energy spectrum of arriving particles, etc. Each sector (a given surface and zone) may belong to one of the non-diagnostic subclasses (this is only a logical restriction and is unenforced), as well as to one or more diagnostic groups. Sectors may also be defined solely to serve as diagnostic surfaces.

The *strata* and *sector_strata_segment* arrays provide labels for coordinating input and output data with external codes. The intention is to use them in the same way as the *zn_index* for individual zones. For historical reasons, a single stratum is used to for sectors on both sides of a boundary (e.g., plasma-surface interface).

nsectors Total number of sectors in the problem.

strata_{sector} External (arbitrary) label for a group of sectors. In the nomenclature of the original DEGAS, one stratum would correspond to a single wall.

sector_strata_segment_{sector} Another external label intended to enumerate individual sectors in a stratum; the original DEGAS analog would be wall segments.

sectors_{pointer} Array pointed to by *surface_sectors* (see *geometry.web*) via the integer index *pointer*. The array *sectors* provides the sector(s) index *sector* associated with a given surface. Need to use a pointer here rather than a direct array with a *surface* index since there may indeed be more than one *sector* associated with a given *surface*. Nonetheless, because of the fact that a sector is defined by a single (and only one) surface, the number of elements in *sectors* is just *nsectors*.

sector_zone_{sector} Returns the zone (class *zn*) associated with *sector*.

sector_surface_{sector} Returns the surface (see *geomint.hweb*) associated with *sector*.

sector_points_{i, e, sector} Returns the *i*th component of the position vector at one end (i.e., *e* = 0 or 1) of *sector*.

sector_type_pointer_{sector, type} Integer index into the list of the *type* subclass corresponding to *sector* (each sector may belong to more than one subclass).

vacuum subclass

sc_vacuum_num Number of sectors in the vacuum subclass.

vacuum_sector_{vacuum_ind} Returns the sector index corresponding to index *vacuum_ind* in the vacuum subclass list.

plasma subclass

sc_plasma_num Number of sectors in the plasma subclass.

plasma_sector_{plasma_ind} Returns the sector index corresponding to index *plasma_ind* in the plasma subclass list.

plasma_e_ion_delta_{plasma_ind} (OBSOLETE. This and the other two *plasma_e_ion* parameters have been replaced by similarly named miscellaneous source parameters. Retaining only this documentation for archival reference.) Contains δ (eV), the constant (i.e., not a function of the incident ion parameters) contribution to the incident ion energy following acceleration through the sheath: $E_{\text{ion}} = \delta + \alpha E_{i,0} + \phi_s Z_i$.

plasma_e_ion_mult_{plasma_ind} Contains α , the multiplier on the sampled ion's energy, $E_{i,0}$, in the expression for the incident ion energy following acceleration through the sheath: $E_{\text{ion}} = \delta + \alpha E_{i,0} + \phi_s Z_i$.

plasma_e_ion_sheath_{plasma_ind} Contains ϕ_s (eV), the sheath potential contribution to the incident ion energy; this gets multiplied by the sampled ions charge Z_i : $E_{\text{ion}} = \delta + \alpha E_{i,0} + \phi_s Z_i$.

target subclass

sc_target_num Number of sectors in the target subclass.

target_sector_{target_ind} Returns the sector index corresponding to index *target_ind* in the target subclass list.

target_material_{target_ind} Integer index into the materials class *ma* to be used to identify the plasma-material interactions for the target sector *target_ind*.

target_temperature_{target_ind} Temperature, in joules, of the target material comprising the target sector *target_ind*.

target_recyc_coef_{target_ind} Recycling coefficient (1 - absorbed fraction) of the target material comprising the target sector *target_ind*.

wall subclass

sc_wall_num Number of sectors in the wall subclass.

wall_sector_{wall_ind} Returns the sector index corresponding to index *wall_ind* in the wall subclass list.

wall_material_{wall_ind} Integer index into the materials class *ma* to be used to identify the plasma-material interactions for the wall sector *wall_ind*.

wall_temperature_{wall_ind} Temperature, in joules, of the wall material comprising the wall sector *wall_ind*.

wall_recyc_coef_{wall_ind} Recycling coefficient (1 - absorbed fraction) of the wall material comprising the wall sector *wall_ind*.

exit subclass

sc_exit_num Number of sectors in the exit subclass.

exit_sector_{exit_ind} Returns the sector index corresponding to index *exit_ind* in the exit subclass list.

diagnostic subclass

sc_diagnostic_grps Number of *groups* of diagnostics in this subclass.

sc_diag_max_bins Used in conjunction with *de_max_bins* to check the dimension for the bins scoring array, *max_bins*.

diagnostic_grp_name_{group_ind} Descriptive label for each diagnostic group.

diagnostic_num_sectors_{group_ind} Number of sectors comprising each diagnostic group.

diagnostic_var_{group_ind} Independent variable for spectrum-collecting (e.g., *sc_diag_energy* or incident angle, *sc_diag_angle*) diagnostics.

diagnostic_tab_index_{group_ind} Number of bins for spectrum-collecting diagnostics. For simplicity, only rank 0 (designated by *diagnostic_tab_index* = 0) or 1 diagnostics are permitted.

diagnostic_min_{group_ind} Minimum value for the independent variable.

diagnostic_delta_{group_ind} Increment between independent variable bins.

diagnostic_spacing_{group_ind} Spacing method for independent variables. Presently only uniform-linear and uniform-log are treated.

diagnostic_grp_base_{group_ind} Index into the list of all sectors for the first member of *groud.ind* (members of each group are required to be consecutive in this list). This array is used only to set up the *diagnostic_sector* macro (see below).

sc_diag_size Total number of diagnostic sectors.

diagnostic_sector_tab_{diagnostic_ind} Sector number corresponding to the diagnostic number *diagnostic_ind*.

Again, this is used only to set up the *diagnostic_sector* macro.

[/Users/dstotler/degas2/src/sector.hweb] Routines using sectors.

sc_check(sector) Verifies that this is a valid sector.

sc_compare(sect, zone, face, stratum, seg) Compares the input zone number *zone*, surface number *face*, stratum label *stratum*, and segment number *seg* with the corresponding properties of sector *sect*. If all four match identically, this macro evaluates as “true”.

sc_plasma_check(p_sector) Verifies that *p_sector* is a valid identifier for a member of the plasma subclass (i.e., *p_sector* is *not* a sector number).

sc_vacuum_check(v_sector) Verifies that *v_sector* is a valid identifier for a member of the vacuum subclass (i.e., *v_sector* is *not* a sector number).

sc_target_check(t_sector) Verifies that *t_sector* is a valid identifier for a member of the target subclass (i.e., *t_sector* is *not* a sector number).

sc_wall_check(w_sector) Verifies that *w_sector* is a valid identifier for a member of the wall subclass (i.e., *w_sector* is *not* a sector number).

sc_exit_check(e_sector) Verifies that *e_sector* is a valid identifier for a member of the exit subclass (i.e., *e_sector* is *not* a sector number).

sc_diag_check(d_sector, grp) Verifies that *d_sector* is a valid identifier for a member of the diagnostic subclass in diagnostic group *grp* (i.e., *d_sector* is *not* a sector number).

diag_lookup(name) Returns the integral diagnostic group number associated with *name*.

plasma_lookup(sector) Retrieves the index in the plasma subclass list corresponding to sector.

target_lookup(sector) Retrieves the index in the target subclass list corresponding to sector.

wall_lookup(sector) Retrieves the index in the wall subclass list corresponding to sector.

sc_diagnostic(group) Provides the integral sector type (analogous to *sc_wall*, *sc_plasma*, etc.) associated with *group*.

diagnostic_sector(i, group) Sector number corresponding to the *i*th member of diagnostic group *group*.

define_sector_vacuum(sector) Adds *sector* to the vacuum sector subclass.

define_sector_plasma(sector, temp) Adds *sector* to the plasma sector subclass. The *temp* property is no longer used, but retained in the macro for compatibility.

define_sector_target(sector, material, temp, recyc) Adds *sector* to the target sector subclass. It is assumed to be made of *material* (member of the material class *ma*) at temperature *temp*, with recycling coefficient *recyc*.

define_sector_wall(sector, material, temp, recyc) Adds *sector* to the wall sector subclass. It is assumed to be made of *material* (member of the material class *ma*) at temperature *temp*, with recycling coefficient *recyc*.

define_sector_exit(sector) Adds *sector* to the exit sector subclass.

28 Detector definitions

[/Users/dstotler/degas2/src/detector.hweb]

```
$Id: 19bb33f2d8ee90d8c81613afab4bdd7580546f80 $
```

[/Users/dstotler/degas2/src/detector.hweb] Parameters and characteristics of detectors. Some of these arrays represent input data, some are derived. These detectors are specifically “viewing” detectors for which a line-of-sight (or “view”) is required. Surface detectors are intended to be represented by the diagnostic sub-class of sectors.

```
"classes.f" 28.1 ≡
@f de_decl integer
@m de_args(x) x
@m de_dummy(x) x
@m de_decl(x) integer x
@m de_lookup(name) string_lookup(name, detector_name1, de_grps)
```

[/Users/dstotler/degas2/src/detector.hweb] Length specifications.

```
"classes.f" 28.2 ≡
@m de_sy_len 24
@m de_name_len 40
@m de_var_unknown 0 // Independent variables
@m de_var_wavelength 1
@m de_spacing_unknown 0 // Spacing for independent variables
@m de_spacing_linear 1
@m de_spacing_log 2
@m de_view_start 0
@m de_view_end 1
@m de_algorithm_unknown 0
@m de_algorithm_uniform 1
@m de_algorithm_circular 2
@m de_view_pointer(i, grp) de_view_tabde_view_base_grp + i - 1
```

[/Users/dstotler/degas2/src/detector.hweb] Definition of the common blocks.

```
"classes.f" 28.3 ≡
  package_init(de)
    define_dimen_pk(de, de_symbol_string, de_sy_len)
    define_dimen_pk(de, de_name_string, de_name_len)

    define_var_pk(de, de_grps, INT)
    define_dimen_pk(de, de_grp_ind, 0, de_grps)

    define_var_pk(de, de_max_bins, INT)

    define_var_pk(de, de_zone_frags_dim, INT)
    define_dimen_pk(de, de_zone_frags_ind, de_zone_frags_dim)
    define_var_pk(de, de_zone_frags_size, INT)

    define_varp_pk(de, detector_name, CHAR, de_name_string, de_grp_ind)
    define_varp_pk(de, detector_num_views, INT, de_grp_ind)

    define_varp_pk(de, detector_var, INT, de_grp_ind)

    define_varp_pk(de, detector_tab_index, INT, de_grp_ind)
    define_varp_pk(de, detector_min, FLOAT, de_grp_ind)
    define_varp_pk(de, detector_delta, FLOAT, de_grp_ind)
    define_varp_pk(de, detector_spacing, INT, de_grp_ind)

    define_var_pk(de, detector_total_views, INT)
    define_dimen_pk(de, de_tot_view_ind, 0, detector_total_views)

    define_dimen_pk(de, de_start_end_ind, de_view_start, de_view_end)
    define_varp_pk(de, de_view_points, FLOAT, vector, de_start_end_ind, de_tot_view_ind)
    define_varp_pk(de, de_view_algorithm, INT, de_tot_view_ind)
    define_varp_pk(de, de_view_halfwidth, FLOAT, de_tot_view_ind)

    define_varp_pk(de, de_zone_frags, FLOAT, de_zone_frags_ind)
    define_varp_pk(de, de_zone_frags_start, INT, de_tot_view_ind)
    define_varp_pk(de, de_zone_frags_num, INT, de_tot_view_ind)
    define_varp_pk(de, de_zone_frags_zones, INT, de_zone_frags_ind)
    define_varp_pk(de, de_zone_frags_min_zn, INT, de_tot_view_ind)
    define_varp_pk(de, de_zone_frags_max_zn, INT, de_tot_view_ind)

    define_varp_pk(de, de_view_base, INT, de_grp_ind)
    define_var_pk(de, de_view_size, INT)
    define_dimen_pk(de, de_view_ind, 0, de_view_size - 1)
    define_varp_pk(de, de_view_tab, INT, de_view_ind)
  package_end(de)
```

29 Detector class attribute descriptions

[/Users/dstotler/degas2/src/detector.hweb]

[/Users/dstotler/degas2/src/detector.hweb] Define detectors. Prefix is *de*.

This class is primarily intended to model light gathering diagnostics of various types, although other uses will probably be found. An individual detector is basically represented by a viewing chord and an angular halfwidth. The signal collected by each detector may (or may not) be broken down into individual bins (e.g., of wavelength).

Each viewing chord, angular halfwidth, and averaging algorithm (i.e., some model for how the physical detector averages the signal over its spot size), together make for a single “view”. One or more related views form a “group”. Each view and group are represented by integers indexing into the arrays below.

de_grps Number of groups which have been defined.

de_max_bins Used in conjunction with *sc_diag_max_bins* to check the dimension for the bins scoring array, *max_bins*.

de_zone_frags_dim Dimension of the compressed *de_zone_frags* array.

de_zone_frags_size Actual size of the compressed *de_zone_frags* array.

detector_name_group_ind Descriptive name for each detector group.

detector_num_views_group_ind Number of views comprising each detector group.

detector_var_group_ind Independent variable for spectrum-collecting (e.g., *de_var_wavelength*) detector groups.

detector_tab_index_group_ind Number of bins for spectrum-collecting detectors. For simplicity, only rank 0 (designated by *detector_tab_index* = 0) or 1 detectors are permitted.

detector_min_group_ind Minimum value for the independent variable.

detector_delta_group_ind Increment between independent variable bins.

detector_spacing_group_ind Spacing method for independent variables. Presently only uniform-linear and uniform-log are treated.

detector_total_views Total number of views which have been defined.

de_view_points_{i, se, view_ind} *i*th position coordinate of the start *se* = *de_view_start* or end *de_view_end* of viewing chord *view_ind*.

de_view_algorithm_{view_ind} Algorithm (currently *de_algorithm_uniform* or *de_uniform_circular*) for averaging over the detector viewing width. See subr. *detector_view_setup* for implementation details.

de_view_halfwidth_{view_ind} Angular half-width (in radians) associated with the viewing chord *view_ind*.

de_zone_frags_{zone_frags_ind} Compressed array containing the path lengths of chords through all of the zones in the problem divided by the zone volume times 4π ; an additional factor accounts for the averaging over a finite halfwidth. Only nonzero values are retained in this array; the subsequent arrays map these data into the full set of zones. This is defined before the run so the chord-integrated contributions can be made from volume scores (during or after the run).

de_zone_frags_start_{view_ind} Pointer into the *de_zone_frags* array indicating the beginning of the list of entries for chord *view_ind*.

de_zone_frags_num_{view_ind} Number of entries in the *de_zone_frags* array for chord *view_ind*.

de_zone_frags_zones_{zone_frags_ind} Zone numbers corresponding to each of the entries in *de_zone_frags*.

de_zone frags_min_zn_{view_ind} Smallest zone number in *de_zone frags_zone* for chord *view_ind*.

de_zone frags_max_zn_{view_ind} Largest zone number in *de_zone frags_zone* for chord *view_ind*.

de_view_base_{group_ind} Index into *de_view_tab* for the first viewing chord of *group_ind*.

de_view_size Length of *de_view_tab*. Note: should be set = 1 if there are no detectors to ensure *de_view_tab* can be allocated and written to the netCDF file.

de_view_tab_i List coordinating the *de_grps* groups with their constituent *detector_total_views* viewing chords.

The *detector_num_views* chords for each group are added to this array when the group is defined with the first one appearing at $i = \text{detector_view_base}_{\text{group_ind}}$.

[/Users/dstotler/degas2/src/detector.hweb] Routines using detectors.

de_lookup(name) Returns the integral detector group number associated with *name*.

de_view_pointer(i, group_ind) *i*th viewing chord ($1 \rightarrow \text{detector_num_views}_{\text{group_ind}}$) of group *group_ind*; i.e., this is the entry into the list of all *detector_total_views* viewing chords.

30 Tally definitions

[/Users/dstotler/degas2/src/tally.hweb]

\$Id: 785bb4ac5e447eef193a254ffcc83630c27a57ca \$

[/Users/dstotler/degas2/src/tally.hweb] Scoring can happen at one of five points within DEGAS 2: (a) when a track crosses a sector (e.g., to compute a flux), (b) when a track travels through a zone (e.g., to compute a track length estimator), (c) at a collision event (e.g., to compute a collision estimator), (d) at a time boundary (e.g., a snapshot estimator of the density), and (e) after the run (to compute a derived score).

Used the word “tally” here instead of “score” since the two-letter prefix for the latter would be too similar to those for sectors and sources.

For further documentation, see the classes.web file.

```
"classes.f" 30.1 ≡
@f tl_decl integer
@m tl_args(x) x
@m tl_dummy(x) x
@m tl_decl(x) integer x
@m tl_copy(x,y) y = x
@m tl_check(x) check_tally(tl_args(x))
```

```
[/Users/dstotler/degas2/src/tally.hweb] Constant specifications.

"classes.f" 30.2 ≡
@m tl_type_max 3 // Types of different tallies
@m tl_type_undefined 0
@m tl_type_sector 1
@m tl_type_test 2
@m tl_type_reaction 3

@m tl_est_unknown 0 // Different estimators
@m tl_est_track 1
@m tl_est_collision 2
@m tl_est_post_process 3
@m tl_est_snapshot 4
@m tl_est_max 4

@m tl_rank_max 5
```

[/Users/dstotler/degas2/src/tally.hweb] Indices for the independent variables. These are associated with strings in subroutine *set_var_list*. Additions here must be made there as well.

```
"classes.f" 30.3 ≡
@m tl_index_unknown 0
@m tl_index_zone 1
@m tl_index_plasma_zone 2
@m tl_index_test 3
@m tl_index_problem_sp 4
@m tl_index_detector 5
@m tl_index_test_author 6 // Reaction or PMI responsible for  $\vec{v}$ 
@m tl_index_reaction 7
@m tl_index_pmi 8
@m tl_index_material 9
@m tl_index_source_group 10 // A source group. Probably won't be used.
@m tl_index_sector 11
@m tl_index_strata 12
@m tl_index_strata_segment 13
@m tl_index_energy_bin 14
@m tl_index_angle_bin 15
@m tl_index_wavelength_bin 16
@m tl_index_diagnostic 17
@m tl_index_zone_ind_1 18
@m tl_index_zone_ind_2 19
@m tl_index_max 100 // An arbitrary limit

@m tl_name_length 80 // For tally names
@m tl_tag_length 40 // For dependent variable names

@m tl_geom_unknown 0 // Tally geometries
@m tl_geom_volume 1
@m tl_geom_surface 2
@m tl_geom_detector 3
@m tl_geom_global 4

@m tl_dep_var_scalar 1 // For tally_dep_var_dim
@m tl_dep_var_vector 3

@m tl_scoring_index(t, j, ip) pr_var_problem_sp_index(t + j - 1, ip tl_index_problem_sp)

@m tl_cv_unknown 0 // Tally conversions
@m tl_cv_scaler_unknown 0
@m tl_cv_partner_unknown 0

@m tl_cv_max_conversions 5 // Fixed dimensioning parameters
@m tl_cv_max_scalers 3
@m tl_cv_max_partners 3
@m tl_cv_max_local_data 5

@m tl_cv_track 1 // Types of conversions
@m tl_cv_post 2
@m tl_cv_output 3

@m tl_cv_scale 1 // Action performed by the conversion
@m tl_cv_to_external_coords 2
@m tl_cv_to_internal_coords 3
@m tl_cv_divide_number 4
```

```
@m tl_cv_test_mass 1 // Scalers and other parameters used
@m tl_cv_problem_sp_mass 2
@m tl_cv_volume 3
@m tl_cv_Pa_per_mTorr 4
@m tl_cv_pos_1 5
@m tl_cv_pos_2 6
@m tl_cv_pos_3 7
@m tl_cv_zone_pos_1 8
@m tl_cv_zone_pos_2 9
@m tl_cv_zone_pos_3 10
@m tl_cv_three_halves 11
@m tl_cv_max_params 11
@m scoring_data_max 200 // Length of scoring_data array
@m max_bins 200
```

[/Users/dstotler/degas2/src/tally.hweb] Variable definitions. *tally_ind* is dimensioned from zero to accomodate netcdf. The 0 index is unused. ??? Start from 1 for now.

```
"classes.f" 30.4 ≡
  package_init(tl)

  define_dimen_pk(tl, tally_type_ind, tl_type_max)
  define_dimen_pk(tl, tally_rank_ind, tl_rank_max)
  define_dimen_pk(tl, tally_est_ind, tl_est_max)
  define_dimen_pk(tl, tally_reac_ind, pr_reaction_dim + so_type_num)
  define_dimen_pk(tl, tally_name_string, tl_name_length)
  define_dimen_pk(tl, tally_tag_string, tl_tag_length)
  define_dimen_pk(tl, tally_cv_ptr_ind, tl_cv_max_conversions)
  define_dimen_pk(tl, tally_cv_scaler_ind, tl_cv_max_scalers)
  define_dimen_pk(tl, tally_cv_partner_ind, tl_cv_max_partners)
  define_dimen_pk(tl, tally_index_ind, 0, tl_index_max)

  define_var_pk(tl, tl_num, INT)
  define_dimen_pk(tl, tally_ind, tl_num)

  define_var_pk(tl, nconversions, INT)
  define_dimen_pk(tl, tally_cv_ind, nconversions)

  define_var_pk(tl, tally_size, INT)

  define_varp_pk(tl, tally_type_num, INT, tally_type_ind)
  define_varp_pk(tl, tally_type, INT, tally_ind)
  define_varp_pk(tl, tally_geometry, INT, tally_ind)
  define_varp_pk(tl, tally_geometry_ptr, INT, tally_ind)
  define_varp_pk(tl, tally_base, INT, tally_ind)
  define_varp_pk(tl, tally_type_base, INT, tally_type_ind)
  define_varp_pk(tl, tally_rank, INT, tally_ind)
  define_varp_pk(tl, tally_dep_var_dim, INT, tally_ind)
  define_varp_pk(tl, tally_indep_var, INT, tally_rank_ind, tally_ind)
  define_varp_pk(tl, tally_tab_index, INT, tally_rank_ind, tally_ind)
  define_varp_pk(tl, tally_name, CHAR, tally_name_string, tally_ind)
  define_varp_pk(tl, tally_dep_var, INT, tally_ind)

  define_varp_pk(tl, tally_est_test, FLOAT, tally_est_ind, tally_ind)
  define_varp_pk(tl, tally_est_reaction, FLOAT, tally_reac_ind, tally_est_ind, tally_ind)

  define_varp_pk(tl, tally_num_conversions, INT, tally_ind)
  define_varp_pk(tl, tally_cv_ptr, INT, tally_cv_ptr_ind, tally_ind)
  define_varp_pk(tl, tally_cv_action, INT, tally_cv_ind)
  define_varp_pk(tl, tally_cv_type, INT, tally_cv_ind)
  define_varp_pk(tl, tally_cv_num_partners, INT, tally_cv_ind)
  define_varp_pk(tl, tally_cv_scalers, INT, tally_cv_scaler_ind, tally_cv_ind)
  define_varp_pk(tl, tally_cv_partners, INT, tally_cv_partner_ind, tally_cv_ind)
  define_var_pk(tl, tally_var_list, CHAR, tally_tag_string, tally_index_ind)

  define_varlocal_pk(tl, tally_version, CHAR, string)
  package_end(tl)
```

[/Users/dstotler/degas2/src/tally.hweb] Common blocks definitions.

```
"classes.f" 30.5 ≡
@f tl_decls integer
@m tl_decls logical check_tally
integer inc
```

[/Users/dstotler/degas2/src/tally.hweb] Setting up tally information.

/* The *ragged_alloc* macros are similar to those in *pmidata.hweb*. More detailed documentation is given there. */

```
"classes.f" 30.6 ≡
@m tl_set_baseinc(x,y,subs) x##_base##subs## = x##_size
    inc = 1 $DO (I, range_min(ind##y##_1), range_max(ind##y##_1))
    {
        inc = inc * y##subs##_I
    }
    inc = x##_dep_var_dim##subs * inc

@m tl_set_base_size(x,y,subs) tl_set_baseinc(x, y, subs)
    x##_size = x##_size + inc
```

31 Tally class attribute descriptions

[/Users/dstotler/degas2/src/tally.hweb]

[*/Users/dstotler/degas2/src/tally.hweb*] Define tallies or scores. Prefix is *tl*.

This class provides the machinery for tabulating the Monte Carlo scores or tallies which form the basis of the DEGAS 2 output. The actual output is controlled within the “output class” (see below).

There are a total of *tl_num* tallies, each representing an accounting of how a physical quantity, the “dependent variable”, depends on a limited set of problem parameters, the “independent variables”. The rank of each tally is in principle arbitrary. Sums over an individual index could either be carried out after the run, or specified as a separate tally (e.g., in order to obtain an explicit estimate of the relative standard deviation in the reduced tally).

For each tally, an estimate of the mean and the relative standard deviation are provided. The latter is ill-defined for tallies with non-positive tally contributions.

There are three types of tally determined by the dependent variable:

tl_type_sector A simple event score carried out when a flight crosses a sector. This can be either a material surface or a purely diagnostic sector (see the sector class above).

tl_type_test A tally based solely on the properties of the test flight itself. The most familiar example is the neutral density.

tl_type_reaction Tallies associated with reactions and sources. The most important examples are the sources of background particles, momenta, and energy arising from each reaction in the problem. The same set of tallies apply to sources. Because this capability was added later, and for the sake of brevity, the associated variable names refer only to “reactions”.

In general, we envision each tally consisting of additive contributions from one or more estimators. At this time, there are only two run time estimators, *tl.est.track* and *tl.est.collision*. The latter is really a catch-all category since it includes not only collision based scores of type *tl.type.test*, but also event-based scores such as those of type *tl.type.sector* and of *tl.type.reaction* which can only be carried out when the reaction actually occurs or in the case of sources. The third type of estimator *tl.est.post_process* must be derived from existing tallies. This option is implemented presently for only certain reactions, sources and tallies; no effort is made to propagate the standard deviation values through this process. A fourth, *tl.est.snapshot*, scores *tl.type.test* tallies at the end of a time dependent calculation. That is, it essentially computes a tally based on the particle positions and velocities as they reach *so_time_final*. There is also a null estimator, *tl.est.unknown* that provides an easy way to ignore the contribution of, say, a particular reaction to a score.

The relative contributions of each estimator to a tally are controlled by the arrays *tally.est.test* (for type *tl.type.est*) and *tally.est.reaction* (*tl.type.reaction*). Since there is only one sensible estimator for *tl.type.sector*, no analogous array is carried along for that type. These arrays contain the multiplicative factor associated with each estimator for that tally. *tally.est.reaction* has an additional index so that different estimator combinations may be specified for each reaction. **Presently, this multiplicative factor is constrained to be either 0 or 1 with the sum over estimators equalling 1.** Once more detailed optimization is carried out, this constraint will be relaxed.

Certain simple manipulations of the tally data can be effected by *conversions*. The principal examples are scaling a tally by a constant (e.g., species mass or zone volume) and velocity coordinate conversion (may be required for coupling DEGAS 2 to a plasma code). The conversions are implemented within the tally class because of the tight links between the two sets of “objects”. The decision not to establish them as separate classes was motivated in part by the treatment of the geometry transformations. Each tally may have associated with it 0 or more (up to *tl_cv_max_conversions*) conversions. Each conversion maintains pointers back to its tally so that the conversions can be carried out completely either by enumerating the conversions for each tally (as is done during tracking) or vice-versa (the case for post-processing). The converted data replace the original so that dimension reducing conversions could not be done with the present approach.

Conversions are carried out at three points during the run; each is designated as a different conversion “type”: *tl_cv_track* for those performed during tracking (the only nontrivial example at present is the conversion of velocities from the internal cartesian coordinates to the external cylindrical system associated with a toroidally symmetric geometry), *tl_cv_post* for conversions needed to prepare the neutral particle number and velocity which are input to the post processing estimator, and *tl_cv_output* for the bulk of the conversions done just before writing the output data.

Presently there are four “actions” performed by the conversions. Although there are no a priori links between the action and the type of conversion, only certain combinations are used in practice; see *tallysetup.web* for specifics. The actions are: *tl_cv_scale* to scale a tally by a constant specified as part of the conversion, *tl_cv_divide_number* to normalize a tally by the number of test particles in a zone; *tl_cv_to_external_coords* and *tl_cv_to_internal_coords* do the velocity conversions. Note, however, that presently *tl_cv_divide_number* does not work as intended for multiple source group runs; for this reason, its use is not recommended.

tl_num Total number of distinct tallies in the problem.

nconversions Total number of distinct conversions in the problem.

tally_size The total size of the array holding all of the tally data.

tally_type_num_{type.ind} Number of tallies of type *type.ind*.

tally_type_{tally.ind} Type of tally *tally.ind*.

tally_geometry_tally_ind Geometry of tally *tally_ind*:

1. *tl_geom_volume* tallies compiled as a function of zone,
2. *tl_geom_surface tl_type_sector* tallies,
3. *tl_geom_detector* tallies computed for a member of the detector class,
4. *tl_geom_global* globally integrated tallies (not yet implemented).

tally_geometry_ptr_tally_ind Points to the geometric group of elements corresponding to tally *tally_ind*. This currently nontrivial only for *tl_geom_surface* (points to a diagnostic group) and *tl_geom_detector* (points to a detector group).

tally_base_tally_ind Pointer to the first datum for tally *tally_ind* in the array holding all of the tally data.

tally_type_base_type_ind Pointer to the first datum of type *type_ind* in the array holding all of the tally data. Note that the tallies of each type are assumed to be consecutive in that array.

tally_rank_tally_ind Number of independent variables (dimensionality) of tally *tally_ind*.

tally_dep_var_dim_tally_ind Dimensionality of dependent variables of tally *tally_ind*; presently either 1 (*tl_dep_var_scalar*) or 3 (*tl_dep_var_vector*).

tally_indep_var_rank_ind, tally_ind Integer index corresponding to the *rank_ind*th independent variable for tally *tally_ind*. The list of possible values is lengthy and probably will be subject to considerable change; refer to *tally.hweb* for the current list (these are macros beginning with *tl_index*; they are matched with strings in subroutine *set_var_list*).

tally_tab_index_rank_ind, tally_ind Number of values of the *rank_ind*th independent variable of tally *tally_ind*.

tally_name_tally_ind Descriptive name for tally *tally_ind*; if the tally is for a vector quantity, the name *must* contain the word “vector”.

tally_dep_var_tally_ind Integer identifying the physical quantity being accumulated for tally *tally_ind*. This is set using a string specified in the argument list to *define_tally* in *tallysetup*. This string is searched for in the global list of dependent variables, *pr_var0_list*, set in subroutine *init_var0_list*. The familiar examples are:

1. *mass_change* source of mass for a background or test species (denoted by the generic term “problem species”) due to the current test species,
2. *momentum_change_vector* source of vector momentum for a problem species due to the current test species,
3. *energy_change* source of energy for a problem species due to the current test species,
4. *mass* mass of test species,
5. *momentum_vector* momentum vector of the test species,
6. *energy* energy of the test species.

Or, it must be one of the other quantities appearing as reaction handling variables (reactiondata class, *reaction_handling_var0*). The most commonly used example are the H_α emission rates which are named *emission_rate_h_alpha* where $h = H, D$, or T according to the current test species.

tally_est_test_est_ind, tally_ind Multiplier on estimator *est_ind* for tally *tally_ind*; makes sense only for tallies of *tl_type_test*.

tally_est_reaction_{reac_so, est_ind, tally_ind} Multiplier on estimator *est_ind* for tally *tally_ind* when scoring a reaction or source. The index *reac_so* is a concatenated list of the problem reactions (first) and the source types. Makes sense only for tallies of *tl_type_reaction*.

tally_num_conversions_{tally_ind} Number of conversions for tally *tally_ind*; must be $\leq tl_cv_max_conversions$.

tally_cv_ptr_{ptr_ind, tally_ind} The *ptr_ind*th conversion for tally *tally_ind*.

tally_cv_action_{cv_ind} The action to be performed by conversion *cv_ind*.

tally_cv_type_{cv_ind} Type of conversion *cv_ind*.

tally_cv_num_partners_{cv_ind} Number of tallies required to carry out *cv_ind*, must be at least one and not more than *tl_cv_max_partners*.

tally_cv_scalers_{scal_ind, cv_ind} List of scalers (constants) *scal_ind* required to carry out *cv_ind*. There may be 0 to *tl_cv_max_scalers* for each conversion; the number is determined by the action.

tally_cv_partner_{part_ind, cv_ind} The *part_ind*th partner in the list of partners for conversion *cv_ind*. The first partner is always the tally being operated on by the conversion. Additional (unmodified) tallies which are required to effect the conversion are included as other “partners”.

[/Users/dstotler/degas2/src/tally.hweb] Routines for use with tallies.

tl_scoring_index(t, j, ip) Provides the integer index of the globally used compiled list of dependent variables, *scoring_data*, for dimension *j* ($j < tally_dep_var_dim_t$) of tally *t*. *ip* is intended to be the up-to-date *index_parameters* array which is currently used to access the background species index.

32 Output definitions

[/Users/dstotler/degas2/src/output.hweb]

```
$Id: 99b8f4fde408b988c52f72f9f73bc80c106a40d2 $
```

[/Users/dstotler/degas2/src/output.hweb] These are the principle output data from DEGAS 2. These are naturally grouped with the tally data, but aren't needed until the run is complete. Hence, we have collected them separately here.

[/Users/dstotler/degas2/src/output.hweb] Indices for moments in the arrays.

```
"classes.f" 32.2 ≡
@m o_mean 0
@m o_var 1 // variance
@m ou_coupling_mass 1 // Rate of background mass exchange
@m ou_coupling_momentum_1 2 // Rate of background momentum exchange
@m ou_coupling_momentum_2 3 // (by components)
@m ou_coupling_momentum_3 4
@m ou_coupling_energy 5 // Rate of background energy exchange
@m ou_coupling_max 5
@m ou_back_max 10 // In lieu of pointer for pr_background_num
@m out_index_5(ind, t) ind1 + tally_tab_indext,1 * (ind2 + tally_tab_indext,2 * (ind3 + tally_tab_indext,3 *
(ind4 + tally_tab_indext,4 * ind5)))
@m out_array_index(i, ind, t) tally_baset + i - 1 + tally_dep_var_dimt * (out_index_5(ind, t))
```

[/Users/dstotler/degas2/src/output.hweb] Variable definitions.

```
"classes.f" 32.3 ≡
package_init(ou)
define_dimen_pk(ou, output_coupling_ind, ou_coupling_max)
define_dimen_pk(ou, output_moments_ind, o_mean, o_var)
define_dimen_pk(ou, ou_back_ind, pr_background_num)

define_dimen_pk(ou, output_tab_ind, 0, tally_size - 1)
define_dimen_pk(ou, output_grps_ind, so_grps)
define_dimen_pk(ou, output_seed_ind, 0, ran_s - 1)

define_var_pk(ou, output_index_1_min, INT)
define_var_pk(ou, output_index_1_max, INT)
define_var_pk(ou, output_index_2_min, INT)
define_var_pk(ou, output_index_2_max, INT)

define_dimen_pk(ou, output_ind_1, output_index_1_min, output_index_1_max)
define_dimen_pk(ou, output_ind_2, output_index_2_min, output_index_2_max)

define_var_pk(ou, output_checkpoint, INT)

define_varp_pk(ou, output_all, FLOAT, output_moments_ind, output_tab_ind)
define_varp_pk(ou, output_grp, FLOAT, output_moments_ind, output_tab_ind, output_grps_ind)

define_varp_pk(ou, out_post_all, FLOAT, output_moments_ind, output_tab_ind)
define_varp_pk(ou, out_post_grp, FLOAT, output_moments_ind, output_tab_ind, output_grps_ind)
define_varp_pk(ou, output_2D_coupling, FLOAT, output_ind_1, output_ind_2, ou_back_ind,
output_coupling_ind, output_grps_ind)
define_varp_pk(ou, output_weight_grp, FLOAT, output_grps_ind)
define_varp_pk(ou, output_num_flights, INT, output_grps_ind)
define_varp_pk(ou, output_random_seed, INT, output_seed_ind, output_grps_ind)
define_varlocal_pk(ou, output_version, CHAR, string)
define_varlocal_pk(ou, output_old_file, INT)

package_end(ou)
```

33 Output class attribute descriptions

[/Users/dstotler/degas2/src/output.hweb]

[*/Users/dstotler/degas2/src/output.hweb*] Definition of the data in the main output file. Prefix is *ou*.

This class contains various objects needed to record the results of the DEGAS 2 calculation. Presently, these consist of several related large arrays representing components of the total scores. The existing arrays are explicitly of rank 2 or 3. The first of these, indicated by *mom_ind* below, is either *o_mean*, indicating the mean score, or *o_var*, denoting its variance. The *table_ind* is the compressed single dimension array containing the ragged, multidimensional scoring data. The *_grp* arrays have a third index *group_ind* for the source group.

With the addition of checkpointing and restarting options, the *output_grp* array no longer contains relative standard deviations in the *o_var* index. Rather, its contents remain in the “raw” form which allows them to be read back into the code and used for further computation during a restart. For convenience, the relative standard deviations have been copied to the *out_post* arrays, although *these have not been corrected for post-processed additions to the mean values*. Hence, detailed statistical analyses should be restricted to the *output_all* and *output_grp* arrays.

Specific data required for the coupling of DEGAS 2 to two dimensional fluid plasma transport codes are contained in the *output_2D_coupling* array. The 2-D spatial indices are determined from the *zn_index* arrays.

output_checkpoint Is *TRUE* when the data in the output file have been generated from a checkpoint. Output data that have been post-processed (not from a checkpoint) are indicated by *FALSE*.

output_all_{mom_ind, table_ind} Sum over all source groups of tallies computed during the DEGAS 2 run.

output_grp_{mom_ind, table_ind, group_ind} Tallies computed during the DEGAS 2 run for source group *group_ind*.

out_post_all_{mom_ind, table_ind} Sum over all source groups of all DEGAS 2 tallies, including post-processed contributions.

output_grp_{mom_ind, table_ind, group_ind} All DEGAS 2 tallies for source group *group_ind*, including post-processed contributions.

output_index_1_min Minimum of the first index appearing in the *zn_index* data passed to DEGAS 2.

output_index_1_max Maximum of the first index appearing in the *zn_index* data passed to DEGAS 2.

output_index_2_min Minimum of the second index appearing in the *zn_index* data passed to DEGAS 2.

output_index_2_max Maximum of the second index appearing in the *zn_index* data passed to DEGAS 2.

output_2D_coupling_{ix, iy, back_ind, cpl_ind, grp_ind} The *cpl_ind*th variable (ion number, momentum component, or energy source) associated with background particle *back_ind* in zone (*ix, iy*), as specified by *zone_index*, for source group *group_ind*.

output_weight_grp_{group_ind} Total weight of particles flown for source group *group_ind*.

output_num_flights_{group_ind} Total number of flights flown for source group *group_ind*. In most cases, *output_weight_grp* should be numerically equal to *output_num_flights*.

output_random_seed_{seed_ind, group_ind} Random seed (a seed is a set of *ran_s* integers, enumerated by *seed_ind*) to be used for source group *group_ind* if there is a restart. These will all have the same value (namely, the seed at the end of the run) if gaps have not been left in the random number sequence (*so_spaced_seeds = TRUE*).

[/Users/dstotler/degas2/src/output.hweb] Routines using output.

Successive attempts have been made at providing a succinct accessor method to the output arrays. The present approach involves the following macro to specify the single array index into which all of the tallies and their dependencies have been compiled:

out_array_index(i, ind, t) Index for the *i*th component, $i < tally_dep_var_dim_t$, of the tally *t* and *ind*, an array of length *tl.rank_max* containing the indices corresponding to a particular set of independent variable values (between 0 and $tally_tab_index_{r, t}-1$ with $1 \leq r \leq tally_rank_t$).

34 Local arrays and machinery to allow compression of scoring data

[/Users/dstotler/degas2/src/stat.hweb]

\$Id: 138baade67a95efbda11a28fa0009ae93cf892b9 \$

[/Users/dstotler/degas2/src/stat.hweb] Ideally these variables would be defined locally and passed solely through argument lists. However, to permit the scoring arrays to be as small as possible, we need to be able to enlarge them on the fly (in subroutines several levels below that in which they were initially defined). The only convenient means of doing that is by using the *define_varp* machinery and passing the data via common where needed.

```
"classes.f" 34.1 ≡
@m stat_mem_inc_ft 2500
@m stat_mem_inc_frag 15000
@m stat_mem_inc_fin 15000
```

```
[/Users/dstotler/degas2/src/stat.hweb] Variable definitions.

"classes.f" 34.2 ≡
  package_init(sa)

  define_dimen_pk(sa, stat_moments_ind, o_mean, o_var)
  define_var_pk(sa, stat_comp_flt, INT)
  define_var_pk(sa, stat_comp_frag, INT)
  define_var_pk(sa, stat_comp_fin, INT)

  define_var_pk(sa, stat_dim_flt, INT)
  define_var_pk(sa, stat_dim_frag, INT)
  define_var_pk(sa, stat_dim_fin, INT)

  define_dimen_pk(sa, stat_flt_ind, 0, stat_dim_flt - 1)
  define_dimen_pk(sa, stat_frag_ind, 0, stat_dim_frag - 1)
  define_dimen_pk(sa, stat_fin_ind, 0, stat_dim_fin - 1)

  define_var_pk(sa, stat_pf_dim_flt, INT)
  define_var_pk(sa, stat_pf_dim_frag, INT)
  define_var_pk(sa, stat_pf_dim_fin, INT)

  define_dimen_pk(sa, stat_pf_flt_ind, 0, stat_pf_dim_flt - 1)
  define_dimen_pk(sa, stat_pf_frag_ind, 0, stat_pf_dim_frag - 1)
  define_dimen_pk(sa, stat_pf_fin_ind, 0, stat_pf_dim_fin - 1)

  define_var_pk(sa, stat_ps_dim_flt, INT)
  define_var_pk(sa, stat_ps_dim_frag, INT)
  define_var_pk(sa, stat_ps_dim_fin, INT)

  define_dimen_pk(sa, stat_ps_flt_ind, 0, stat_ps_dim_flt - 1)
  define_dimen_pk(sa, stat_ps_frag_ind, 0, stat_ps_dim_frag - 1)
  define_dimen_pk(sa, stat_ps_fin_ind, 0, stat_ps_dim_fin - 1)

  define_var_pk(sa, stat_wt_dim_frag, INT)
  define_var_pk(sa, stat_wt_dim_fin, INT)

  define_dimen_pk(sa, stat_wt_frag_ind, 0, stat_wt_dim_frag - 1)
  define_dimen_pk(sa, stat_wt_fin_ind, 0, stat_wt_dim_fin - 1)

  define_var_pk(sa, stat_size_flt, INT)
  define_var_pk(sa, stat_size_frag, INT)
  define_var_pk(sa, stat_size_fin, INT)

  define_var_pk(sa, stat_wt_tot_flt, FLOAT)
  define_var_pk(sa, stat_wt_tot_frag, FLOAT)
  define_var_pk(sa, stat_wt_tot_fin, FLOAT)

  define_varp_pk(sa, stat_flt, FLOAT, stat_moments_ind, stat_flt_ind)
  define_varp_pk(sa, stat_frag, FLOAT, stat_moments_ind, stat_frag_ind)
  define_varp_pk(sa, stat_fin, FLOAT, stat_moments_ind, stat_fin_ind)

  define_varp_pk(sa, stat_ptr2full_flt, INT, stat_pf_flt_ind)
  define_varp_pk(sa, stat_ptr2full_frag, INT, stat_pf_frag_ind)
  define_varp_pk(sa, stat_ptr2full_fin, INT, stat_pf_fin_ind)

  define_varp_pk(sa, stat_ptr2short_flt, INT, stat_ps_flt_ind)
  define_varp_pk(sa, stat_ptr2short_frag, INT, stat_ps_frag_ind)
  define_varp_pk(sa, stat_ptr2short_fin, INT, stat_ps_fin_ind)
```

```
/* frag and fin are used as a base (and an increment), but flt is always an increment. The weight of
   each entry in a base depends on the scoring history to that point. It's constant for an increment.
 */
define_varp_pk(sa, stat_wt_frag, FLOAT, stat_wt_frag.ind)
define_varp_pk(sa, stat_wt_fin, FLOAT, stat_wt_fin.ind)
package_end(sa)
```

35 Description of compression algorithm and data structures

[/Users/dstotler/degas2/src/stat.hweb]

[/Users/dstotler/degas2/src/stat.hweb] Define structures for compression of scores. Prefix is *sa*.

This class contains the local arrays which carry the scoring data into the output class arrays. A separate class is used to permit these arrays to be compressed (removing entries which are 0), so that operations on the entire arrays are much quicker than those on the full-sized output arrays. At least at the single-flight level, the latter are filled with zeroes apart from the relatively small section of the problem space covered by that flight.

Much of this class deals with the bookkeeping required for managing the compression and decompression process. Furthermore, almost every variable or array appears three times. There are separate “objects” for each level of the code: *flt*, each flight; *frag*, the “fragment” of the total number of flights assigned to each processor; and *fin*, the final set of scores which will be decompressed into the output arrays. The descriptions below will have an appended *fff*, which is intended to stand for these three strings.

Every array below has a corresponding dimension in order to permit full flexibility in enabling and disabling compression at each level. If the data at each level were always compressed, fewer dimension variables would be required. In practice, it is expected that the *flt* data will always be compressed. Since the output class already contains an uncompressed version of the final data, do not expect to want to compress the *fin* data. Hence, *stat.fin* is not allocated and is replaced in the code explicitly by corresponding output class arrays. However, the code required for compression is in place and commented out just in case; e.g., if it is decided that the output arrays should be compressed as well. Whether or not it is best to compress the *frag* data remains to be determined. The optimum choice may depend upon the number of processors.

stat_comp_fff Integer flag (to be set using the *FALSE* and *TRUE* flags) indicating whether or not the data at level *fff* are to be compressed.

stat_dim_fff Integer dimension of *stat_fff* array.

stat_pf_dim_fff Integer dimension of *stat_ptr2full_fff* array.

stat_ps_dim_fff Integer dimension of *stat_ptr2short_fff* array.

stat_wt_dim_fff Integer dimension of *stat_wt_fff* array (none for *flt*; see below).

stat_size_fff Current number of entries in *stat_fff*. If *fff* is not compressed, this is *tally_size*.

stat_wt_tot_fff Total weight represented by the current contents of *stat_fff*.

stat_fff[i_s] Is the *i_s*th nonzero score compiled at level *fff*.

stat_ptr2full_fff[i_s] Pointer to the entry in the full (i.e., uncompressed) array corresponding to the *i_s* nonzero score compiled at level *fff*.

stat_ptr2short_fff[i_f] Pointer from the full (i.e., uncompressed) array index *i_f* back to its corresponding entry in the compressed array *stat_fff*. The guess is that it is more efficient to carry around this array than to search *stat_ptr2full_fff* with each value of *i_f* that comes along during the scoring process.

stat_wt_frag[i_s] Weight associated with the *i_s* nonzero score in *stat_fff*. Note that there is no such array for *flt* since the weight is a constant (usually unity) for all entries.

[/Users/dstotler/degas2/src/stat.hweb] Macros for Stat class.

stat_mem_inc_fff Initial dimension and memory increment for compressed arrays at level *fff*. The values used here have been chosen empirically to minimize the number of run time reallocations.

36 Random definitions

[/Users/dstotler/degas2/src/random.hweb]

\$Id: 96c9580ecfd292e59499a792839c6254460f1ee6 \$

[/Users/dstotler/degas2/src/random.hweb] A random number sequence is identified by a string tag.

```
"classes.f" 36.1 ≡
@f rn_decl integer
@f rn_decls integer
@f rn_seed_decl integer
@f single_precision integer

@m rn_args(x) rn_index(x), rn_array(x)_0
@m rn_dummy(x) rn_index(x), rn_array(x)
@m rn_decl(x) integer rn_index(x)
    real rn_array(x)_0:ran_k-1
@m rn_decls real random
    external random

@m rn_copy(x,y) rn_index(y) = rn_index(x)
    do ran_temp = 0, ran_k - 1
        rn_array(y)_ran_temp = rn_array(x)_ran_temp
    end do

@m rn_index(x) rn_index1(x) // Accessor routines
@m rn_index1(x) ran_index##x
@m rn_array(x) rn_array1(x)
@m rn_array1(x) ran_array##x
```

[/Users/dstotler/degas2/src/random.hweb] Length specifications.

```
"classes.f" 36.2 ≡
@m ran_k 100 // The size of the batch of random numbers
@m ran_s 8 // The size of the seed.
@m ran_c 34 // The size of the decimal version of seed.
```

[/Users/dstotler/degas2/src/random.hweb] Inline calling routines.

```
/* An inline version of  $y = \text{random}(\text{rn\_args}(x))$  */

"classes.f" 36.3 ≡
@m rn_next(y,x)  $y = \text{random}(\text{rn\_args}(x))$ 
@m rn_init(seed,x) call random_init_d2(seed, rn_args(x))
@m rn_seed_copy(x,y) rn_seed_copy1(x, y)
@m rn_seed_copy1(x,y) $DO (I, 0, ran_s - 1)
{
    y_I = x_I
}
@m rn_seed_decl(x) integer x_0:ran_s-1
@m rn_seed_args(x) x_0
@m rn_iso_next(v,x) call random_isodist(v, 1, rn_args(x))
@m rn_cos_next(v,x) call random_cosdist(v, 1, rn_args(x))
@m rn_gauss_next(y,x) call random_gauss(y, 1, rn_args(x))
```

37 Cross section definitions

[/Users/dstotler/degas2/src/xsection.hweb]

\$Id: 4831e4b2192f1184bde6cf6e81fb8cb24c75e72e \$

Information about crosssections.

[/Users/dstotler/degas2/src/xsection.hweb] Constant definitions.

```
"classes.f" 37.1 ≡
@#m xs_spacing_unknown 0 // Types of spacing in TABLES
@#m xs_spacing_linear 1
@#m xs_spacing_log 2

@#m xs_var_unknown 0 // Indices for the independent variables
@#m xs_var_density 1 // Indices for the independent variables
@#m xs_var_energy 2
@#m xs_var_temperature 3
@#m xs_var_sp_energy 4
@#m xs_var_sp_temperature 5
@#m xs_var_zone 6
@#m xs_var_1st_random_number 7
@#m xs_var_v_test_1 8
@#m xs_var_v_test_2 9
@#m xs_var_v_test_3 10
@#m xs_var_v_flow_1 11
@#m xs_var_v_flow_2 12
@#m xs_var_v_flow_3 13
@#m xs_var_m_test 14
@#m xs_var_m_back 15
@#m xs_var_m_emitter 16
@#m xs_var_v_flowb_1 17
@#m xs_var_v_flowb_2 18
@#m xs_var_v_flowb_3 19
@#m xs_var_elec_temperature 20

@#m xs_max_random 1
@#m xs_max_indep_params 20

@#m xs_unit_string_length 12 // The length of the unit specifications
@#m xs_tag_string_length 40 // The length of variable names and spacings
@#m xs_eval_name_length 40 // Routine used to evaluate FITS

@#m xs_table_rank_max 3 // The biggest rank table
@#m xs_dep_var_max 20 // Maximum number of dependent variables
```

[/Users/dstotler/degas2/src/xsection.hweb] Accessor functions for *xs_data_tab* to make it look like a 4-dimensional arrays.

```
"classes.f" 37.2 ≡
@#m xs_data_table(i,j,k,n) xs_data_tabi+xs_tab_indexn,1*(j+xs_tab_indexn,2*k)+xs_data_basen
```

[/Users/dstotler/degas2/src/xsection.hweb] variable definitions.

```
"classes.f" 37.3 ≡
  package_init(xs)
    define_dimen_pk(xs, rank_ind, xs_table_rank_max)
    define_dimen_pk(xs, rank_ind0, 0, xs_table_rank_max)
    define_dimen_pk(xs, dep_var_ind, xs_dep_var_max)
    define_dimen_pk(xs, unit_string, xs_unit_string_length)
    define_dimen_pk(xs, tag_string, xs_tag_string_length)
    define_dimen_pk(xs, xs_symbol_string, rc_sy_len)
    define_dimen_pk(xs, eval_name, xs_eval_name_length)

    /* The "size" variables get used in the table allocation macros. */
    define_var_pk(xs, xs_data_size, INT)
    define_dimen_pk(xs, xs_data_ind, 0, xs_data_size - 1)

    define_var_pk(xs, xs_num_dep_var, INT)
    define_var_pk(xs, xs_rank, INT, dep_var_ind)
    define_var_pk(xs, xs_tab_index, INT, rank.ind, dep_var.ind)
    define_var_pk(xs, xs_data_base, INT, dep_var.ind)
    define_var_pk(xs, xs_data_inc, INT, dep_var.ind)

    define_var_pk(xs, xs_name, CHAR, xs_symbol_string)
    define_var_pk(xs, xs_spacing, CHAR, tag_string, rank.ind0, dep_var.ind)
    define_var_pk(xs, xs_var, CHAR, tag_string, rank.ind0, dep_var.ind)
    define_var_pk(xs, xs_units, CHAR, unit_string, rank.ind0, dep_var.ind)
    define_var_pk(xs, xs_eval_name, CHAR, eval.name, dep_var.ind)

    define_var_pk(xs, xs_min, FLOAT, rank.ind, dep_var.ind)
    define_var_pk(xs, xs_max, FLOAT, rank.ind, dep_var.ind)
    define_var_pk(xs, xs_mult, FLOAT, rank.ind0, dep_var.ind)

    define_var_pk(xs, xs_data_tab, FLOAT, xs_data.ind)
    define_varlocal_pk(xs, xsection_version, CHAR, string)
  package_end(xs)
```

[/Users/dstotler/degas2/src/xsection.hweb] Common blocks definitions.

```
"classes.f" 37.4 ≡
  @f xs_decls integer
  @m xs_decls integer old_size
```

[/Users/dstotler/degas2/src/xsection.hweb] Other macros

```
/* The following macros are merely renamed versions from pmiformat.hweb. Any description may be
found in that file. */

"classes.f" 37.5 ≡
@m set_base(x, inc) $ASSERT(range_min(ind##x##_base_1) ≡
    range_min(ind##inc##_1) ∧ range_max(ind##x##_base_1) ≡
    range_max(ind##inc##_1))x##_base range_min(ind##x##_base_1) = 0 $DO(I,
    range_min(ind##x##_base_1) + 1, range_max(ind##x##_base_1))
{
    x##_base_I = x##_base_{I-1} + inc_{I-1}
}
x##_size = x##_base range_max(ind##x##_base_1) + inc range_max(ind##inc##_1)

@if 1
@m set_inc(x, y) $DO(J, range_min(ind##y##_2), range_max(ind##y##_2)) { x##_inc_J = 1
    $DO(I, range_min(ind##y##_1), range_max(ind##y##_1))
    {
        x##_inc_J = x##_inc_J * y_{J, J}
    }
}
@else
@m set_inc(x, y) do j = range_min(ind##y##_2), range_max(ind##y##_2)
    x##_inc_j = 1
    do i = range_min(ind##y##_1), range_max(ind##y##_1)
        x##_inc_j = x##_inc_j * y_{i, j}
    end do
end do
#endiff
@m xs_ragged_alloc(x, y) set_inc(x, y)
set_base(x, x##_inc)
var_alloc(x##_tab)

@m xs_ragged_realloc(x, y) old_size = x##_size
set_inc(x, y)
set_base(x, x##_inc)
var_realloc(x##_tab, old_size - 1, x##_size - 1)
```

38 Cross section class attribute descriptions

[/Users/dstotler/degas2/src/xsection.hweb]

[/Users/dstotler/degas2/src/xsection.hweb] Format for cross sections and other atomic physics data. Prefix is *xs*.

The properties associated with “cross sections” (which is a generic designation for not only cross sections, but all related quantities such as reaction rates, momentum transfer rates, etc. presented below. Here, the subscript *j* denotes the index for dependent variables (dimension variable *dep_var_ind*), while the nonzero values of the subscript *i* represent the independent variables (dimension variables *rank_ind* and *rank_ind0*) needed to determine a particular value of variable *j*. Some of these variables apply exclusively to “fits”, some to “tables”; some apply to both. *Note: fits are not yet implemented.* See also the documentation in *ratecalc.web*.

xs_data_size Size of one-dimensional array containing all data for this process.

xs_num_dep_var Number of dependent variables associated with reaction *xs_name*.

xs_rank_j Integer number of independent variables for this particular “cross section”.

xs_tab_index_{i, j} For tabular organizations, this integer provides the number of values used for independent variable *i* and dependent variable *j*.

xs_data_base_j Index into 1-D data array for first element associated with *j*.

xs_data_inc_j Number of data elements associated with *j*.

xs_name String naming the “cross section”. Same as the reaction name; that is, it should match *rc-sy(rc_num)*.

xs_spacing_{0, j} String describing the way the data are to be interpolated. Either ‘linear’ or ‘log’; applies to tabular organization only.

xs_spacing_{i, j} String telling the method of interpolation to be used for independent variable *i*; applies to tabular organization only.

xs_var_{0, j} String which tells more precisely what this quantity is; e.g., ‘rate’, ‘cross_section’.

xs_var_{i, j} String describing independent variable *i*. Examples are ‘density’, ‘energy’, ‘temperature’, ‘specific_energy’ (i.e., energy per amu), and ‘specific_temperature’. Note: the “non-specific” forms should only be used for electron-impact processes where there can be no confusion as to the relevant mass.

xs_units_{0, j} String providing the units for this quantity; may be in any system.

xs_units_{i, j} String providing the units for independent variable *i*.

xs_eval_name_j Mechanism for evaluating dependent variable *j*; e.g., *table_calc*, *table_eval*, *table_external*, or *fit* (not implemented yet).

xs_min_{i, j} Floating point minimum value to be used for independent variable *i* in a table. It will be combined with the maximum and the number of values to actually define the array of independent variable values. For a fit, provides the lower range of validity for this independent variable.

xs_max_{i, j} Floating point maximum value to be used for independent variable *i* in a table. For a fit, provides the upper range of validity for this independent variable.

xs_mult_{0, j} Floating point multiplier for this quantity. While it may be used for any necessary purpose, it should provide the conversion factor from *xs_units_{0, j}* to the MKS system.

xs_mult_{i, j} Floating point multiplier for independent variable *i*. It should provide the conversion factor from *xs_units_i* to the MKS system.

xs_data_tab One-dimensional array containing all data values.

[/Users/dstotler/degas2/src/xsection.hweb] Routines using cross section data.

xs_data_table(*i, j, k, n*) Multidimensional accessor function to cross section data provided for easier use; *i, j, k* are the independent variable indices, $0 \leq i \leq xs_tab_index_1, n-1$, etc. for the *n*th dependent variable.

xs_ragged_alloc(*xs_data, xs_tab_index*) Computes sizes and base addresses of the one-dimensional data array *xs_data_tab* (*xs_data* is a root name here) based on the multi-dimensional information provided by *xs_tab_index*. There is additional documentation of a similar macro in *pmiformat.hweb*.

xs_ragged_realloc(*xs_data, xs_tab_index*) Makes changes to size of one-dimensional data array *xs_data_tab* based on a revised *xs_tab_index*. The user must be adding to the end of the table for this to work properly.

39 Rate calculation definitions

[/Users/dstotler/degas2/src/ratecalc.hweb]

\$Id: eedb9aa65ab49e9c988eb07dc8d33137b4955b5b \$

Define a few macros for neatness sake.

[/Users/dstotler/degas2/src/ratecalc.hweb] Parameters used in integration routines

```
"classes.f" 39.1 ≡
@m integration_acc const(1.0, -7) // Fractional accuracy desired
@m integration_infinity const(1.0, 20) // ∞ for upper bound of integrations; should be > 1/epsilon
@m integ_transform_exp 4 // Exponent for power law variable transformation
@m integ_transform_ratio const(2., -2) // Smaller 1/upper bounds get variable transformation
@m min_bound_ratio const(1.003) // Min. ratio of bounds in find_threshold
@m ev_to_ergs (electron_charge * const(1.0, 7)) // To convert eV to ergs
@m atomic_mass_unit_g (atomic_mass_unit * const(1.0, 3)) // 1 amu in grams
@m max_integ_steps 20 // Maximum number of integration steps
@m num_extrap_pts 3 // Points used to extrapolate estimates
@m max_interp_pts 10 // Maximum number of interp. points
@m msg_length 80 // Length of ALADDIN error messages
@m ra_search_label_length 40 // Length of labels used to identify data
@m ra_data_filename_length 80 // Length of filename containing data
@m ra_label_max 100 // Maximum number of search labels for data
@m ra_unit_string_length 12 // Length of unit specifications
@m ra_fit_coef_max 13000 // Maximum number of fit coefficients
```

[/Users/dstotler/degas2/src/ratecalc.hweb] variable definitions

```
"classes.f" 39.2 ≡
  package_init(ra)
  define_dimen_pk(ra, search_label, ra_search_label_length)
  define_dimen_pk(ra, label_num_ind, ra_label_max)
  define_dimen_pk(ra, data_filename, ra_data_filename_length)
  define_dimen_pk(ra, mass_unit_string, ra_unit_string_length)
  define_dimen_pk(ra, ra_dep_var_ind, xs_dep_var_max)
  define_var_pk(ra, ra_mass, FLOAT)
  define_var_pk(ra, ra_mass_units, CHAR, mass_unit_string)
  define_var_pk(ra, ra_num_hier, INT)
  define_var_pk(ra, ra_num_bool_and, INT, ra_dep_var_ind)
  define_var_pk(ra, ra_num_bool_not, INT, ra_dep_var_ind)
  define_var_pk(ra, ra_hier_label, CHAR, search_label, label_num_ind)
  define_var_pk(ra, ra_bool_and_label, CHAR, search_label, label_num_ind, ra_dep_var_ind)
  define_var_pk(ra, ra_bool_not_label, CHAR, search_label, label_num_ind, ra_dep_var_ind)
  define_var_pk(ra, ra_data_filename, CHAR, data_filename)
  package_end(ra)
```

[/Users/dstotler/degas2/src/ratecalc.hweb] Common block definitions

/* ALADDIN common blocks use implicit naming convention; define a macro to take care of the type declarations when ALPCOM.FOR is used. */

```
"classes.f" 39.3 ≡
@f decl_alpcom common
@m decl_alpcom integer nelnmx, nhlmx, nblmx
  integer leseqn, neln, nhl, nbl, lcmpr, ncmln, lcfptr, ncfln
  include"../Aladdin/ALPCOM.FOR"
```

[/Users/dstotler/degas2/src/ratecalc.hweb] These are statement labels for the input reading routines

```
"classes.f" 39.4 ≡
@m next_dep_var #:0
@m end_dep_var #:0
@m next_line #:0
@m end_indep_var #:0
@m react_begin #:0
@m next_hlab #:0
@m bool_sect #:0
@m new_bool_sect #:0
@m bool_list #:0
@m next_bool #:0
@m react_done #:0

/* And a local common block */
@f ra_localcommon integer
@m ra_localcommon real u, up, temperature
double precision fit_coef(ra_fit_coef_max)
integer moment_number, num_fit_coef, num_evaluations
character*xs_eval_name_length leval
common u, up, temperature, fit_coef, moment_number, num_fit_coef, leval, num_evaluations
```

40 Rate calculation class attribute descriptions

[/Users/dstotler/degas2/src/ratecalc.hweb]

[/Users/dstotler/degas2/src/ratecalc.hweb] Data and local commons used in calculation of atomic physics rates. Prefix is *ra*

The variables in this class specify the information required to retrieve data from external database files. Presently, the database is assumed to be ALADDIN. The specific variables used here reflect that assumption. A single file name and set of hierarchical search labels are used for all of the data to be associated with one particular instance of this class. However, as many as *xs_dep_var_max* different (index *j* below) sets of Boolean search labels may be specified. This allows different quantities (e.g., cross sections and reaction rates) to be included here, but forces them to be closely related. In this way, we hope to ensure overall consistency of the data.

ra_mass Mass to be used in relating energy and velocity when evaluating the argument of the function specified by this instance.

ra_mass_units Units of *ra_mass*.

ra_num_hier Number of search labels in array *ra_hier_label*.

ra_num_bool_and_j Number of labels in *ra_bool_and_label*.

ra_num_bool_not_j Number of labels in *ra_bool_not_label*.

ra_hier_label Array of of hierarchical search labels used to identify (e.g., for use with ALADDIN) data.

ra_bool_and_label_j Array of labels used to further identify information associated with dependent variable *j*; in searching the data files, entries which *contain* these strings are selected.

ra_bool_not_label_j Array of labels used to further identify information associated with dependent variable *j*; in searching the data files, entries which *do not contain* these strings are selected.

ra_data_filename File name from which data will be extracted.

41 Plasma-Material Interaction (PMI) format definitions

[/Users/dstotler/degas2/src/pmiformat.hweb]

```
$Id: fba0fb8bda440a6d673f6d35499341447888f04f $
```

[/Users/dstotler/degas2/src/pmiformat.hweb] Information about data used to define plasma-material interactions

[/Users/dstotler/degas2/src/pmiformat.hweb] Constant definitions.

```
"classes.f" 41.2 ≡
@m pf_spacing_unknown 0 // Types of spacing of data and
@m pf_spacing_linear 1 // independent variable arrays
@m pf_spacing_log 2

@m pf_var_unknown 0 // Indices for the independent variables
@m pf_var_energy_in 1 // These double as the indices for the
@m pf_var_vel_1_in 2 // independent_paramaters array used
@m pf_var_vel_2_in 3 // in the main code.
@m pf_var_vel_3_in 4
@m pf_var_polar_angle_in 5
@m pf_var_t_wall 6
@m pf_var_1st_random_number 7 // These must be consecutive!
@m pf_var_2nd_random_number 8
@m pf_var_3rd_random_number 9
@m pf_var_cos_polar_angle_in 10

@m pf_max_random 3
@m pf_max_indep_params 10

@m pf_unit_string_length 12 // The length of the unit specifications
@m pf_tag_string_length 20 // The length of variable names and spacings
@m pf_eval_string_length 40 // Routine used to evaluate data

@m pf_table_rank_max 5 // The biggest rank for data tables
@m pf_dep_var_max 5 // Maximum number of dependent variables
```

[/Users/dstotler/degas2/src/pmiformat.hweb] Accessor functions to make main tables look like multi-dimensional arrays

```
"classes.f" 41.3 ≡
@m pf_data_table(i,j,k,l,m,n)
pf_data_tabi+pf_tab_index_n,1*(j+pf_tab_index_n,2*(k+pf_tab_index_n,3*(l+pf_tab_index_n,4*m)))+pf_data_base_n
```

[/Users/dstotler/degas2/src/pmiformat.hweb] variable definitions.

```
"classes.f" 41.4 ≡
  package_init(pf)

  define_dimen_pk(pf, pf_rank_ind, pf_table_rank_max)
  define_dimen_pk(pf, pf_rank_ind0, 0, pf_table_rank_max)
  define_dimen_pk(pf, pf_dep_var_ind, pf_dep_var_max)
  define_dimen_pk(pf, pf_unit_string, pf_unit_string_length)
  define_dimen_pk(pf, pf_tag_string, pf_tag_string_length)
  define_dimen_pk(pf, pf_symbol_string, pm_sy_len)
  define_dimen_pk(pf, pf_eval_string, pf_eval_string_length)

  /* The "size" variables get used in the table allocation macros. */
  define_var_pk(pf, pf_data_size, INT)
  define_dimen_pk(pf, pf_data_ind, 0, pf_data_size - 1)

  define_var_pk(pf, pf_num_dep_var, INT)
  define_var_pk(pf, pf_rank, INT, pf_dep_var_ind)
  define_var_pk(pf, pf_tab_index, INT, pf_rank_ind, pf_dep_var_ind)
  define_var_pk(pf, pf_data_base, INT, pf_dep_var_ind)
  define_var_pk(pf, pf_data_inc, INT, pf_dep_var_ind)

  define_var_pk(pf, pf_name, CHAR, pf_symbol_string)
  define_var_pk(pf, pf_spacing, CHAR, pf_tag_string, pf_rank_ind0, pf_dep_var_ind)
  define_var_pk(pf, pf_var, CHAR, pf_tag_string, pf_rank_ind0, pf_dep_var_ind)
  define_var_pk(pf, pf_units, CHAR, pf_unit_string, pf_rank_ind0, pf_dep_var_ind)
  define_var_pk(pf, pf_eval_name, CHAR, pf_eval_string, pf_dep_var_ind)

  define_var_pk(pf, pf_min, FLOAT, pf_rank_ind, pf_dep_var_ind)
  define_var_pk(pf, pf_max, FLOAT, pf_rank_ind, pf_dep_var_ind)
  define_var_pk(pf, pf_mult, FLOAT, pf_rank_ind0, pf_dep_var_ind)

  define_var_pk(pf, pf_data_tab, FLOAT, pf_data_ind)
  define_varlocal_pk(pf, pf_version, CHAR, string)
  package_end(pf)
```

[/Users/dstotler/degas2/src/pmiformat.hweb] Common blocks definitions.

```
"classes.f" 41.5 ≡
  @f pf_decls integer
  @m pf_decls integer old_size
```

[/Users/dstotler/degas2/src/pmiformat.hweb] Other macros

```

/* set_base(x, y) is used to set the array x_base(i) which points to the ith slice of the
rank 1 array x_tab. The length of each slice is provided by the input array inc(i) (set up via set_inc). The
total size of x_tab is stored in x_size. */

"classes.f" 41.6 ==
@m set_base(x, inc) $ASSERT(range_min(ind##x##_base_1) ==
range_min(ind##inc##_1) & range_max(ind##x##_base_1) ==
range_max(ind##inc##_1))x##_base range_min(ind##x##_base_1) = 0 $DO(I,
range_min(ind##x##_base_1) + 1, range_max(ind##x##_base_1))

{
x##_base_I = x##_base_{I-1} + inc_{I-1}
}
x##_size = x##_base range_max(ind##x##_base_1) + inc range_max(ind##inc##_1)

/* From the array y(m, n), set_inc computes the number of elements implied in the mth
dimensional object represented by each n and stores it in x_inc(n). The first loop is over the the
dependent variables; the second is over the rank of each dependent variable. */

@m set_inc(x, y) /*
$ASSERT( rank##y == 2)
*/
$DO(J, range_min(ind##y##_2), range_max(ind##y##_2)) { x##_inc_J = 1
$DO(I, range_min(ind##y##_1), range_max(ind##y##_1))
{
x##_inc_J = x##_inc_J * y_{I, J}
} }

/* pf_ragged_alloc automatically sets up and allocates memory for “ragged” arrays used to store
the main tabular data. The first argument is intended to be a root name. A family of arrays
sharing that root name are required for this macro. These are x_base (pointer to the starting
point of individual array slices), x_tab (the 1-D array which will actually be allocated), x_size
(scalar which contains the total number of elements), and x_inc (number of elements in a given
array slice; i.e., the increment). The macro set_inc is called to set x_inc. A separate accessor
macro is used to equivalence x_tab to the apparently multi-rank array seen by the user. y
describes the rank and size of each dependent variable; namely, it will be the array pf_tab_index
defined above. x will be pf_data. */
@m pf_ragged_alloc(x, y) set_inc(x, y)
set_base(x, x##_inc)
var_alloc(x##_tab)

/* pf_ragged_realloc is exactly like pf_ragged_alloc except for saving the old_size and finishing with
a call to var_realloc. Since x_base etc. are recomputed from scratch, the only requirement on the
change in the array dimensions is that the user add on to the end (rather than insert in between
existing slices) of the table. */

@m pf_ragged_realloc(x, y) old_size = x##_size
set_inc(x, y)
set_base(x, x##_inc)
var_realloc(x##_tab, old_size - 1, x##_size - 1)

```

42 PMI format class attribute descriptions

[/Users/dstotler/degas2/src/pmiformat.hweb]

[/Users/dstotler/degas2/src/pmiformat.hweb] Define structures used to specify plasma-material interaction data. Prefix is *pf*.

This class provides a format for external storage of data which deal with a particular PMI. Here, the subscript *j* denotes the index for dependent variables, while the nonzero values of the subscript *i* represent the independent variables needed to determine a particular value of variable *j*.

pf-data-size Size of one-dimensional array containing all data for this PMI.

pf-num-dep-var Number of dependent variables associated with this PMI.

pf-rank_j Integer number of independent variables for the *j*th dependent variable.

pf-tab-index_{j,i} Integer number of values used for the *i*th independent variable of the *j*th dependent variable.

pf-data-base_j Index in *pf-data-tab* at which the data for the *j*th dependent variable begin. This array is set by the macros associated with *pf-ragged-alloc* and *pf-ragged-realloc*.

pf-data-inc_j Is the number of elements in *pf-data-table* required for the *j*th dependent variable. This array is set by the macros associated with *pf-ragged-alloc* and *pf-ragged-realloc*.

pf-name String (length *pm-sy-len*) naming the PMI. Same as the name of the PMI in the *pm* class; that is, it should match *pm-sy*.

pf-spacing_{j,i} String (length *pf-tag-string-length*) describing the way the data are to be interpolated. Examples are ‘linear’, ‘log’, and ‘nonuniform’; applies to tabular data only.

pf-var_{j,i} String (length *pf-tag-string-length*) describing the *i*th independent variable of the *j*th dependent variable. *i* = 0 refers to the dependent variable itself. Examples are ‘yield’, ‘energy-in’, etc.; see *pmiformat.hweb* for a complete list.

pf-units_{j,i} String (length *pf-eval-string-length*) providing the units for the *i*th independent variable *i*. String providing the units for this quantity; may be in any system.

pf-eval-name_j String (length *pf-eval-string-length*) identifying the means for evaluating the *j*th dependent variable. The most prevalent value is “table”; other values must correspond to the name of a fitting subroutine. See *plate.web* and the description of the *pd* class for more details. Note that in some instances, the appropriate dependent “variable” is really a vector (i.e., velocity); in this case, tabular evaluation (which can return only a single scalar) makes no sense.

pf-min_{j,i} Floating point minimum value to be used for the *i*th independent variable of the *j*th dependent variable. It will be combined with the maximum and the number of values to actually define the array of independent variable values.

pf-max_{j,i} Floating point maximum value to be used for the *i*th independent of the *j*th dependent variable.

pf-mult_{j,i} Floating point multiplier for the *i*th independent variable of the *j*th dependent variable. *i* = 0 refers to the dependent variable itself. It should provide the conversion factor from *pf-units_{j,i}* to the MKS system.

pf-data-tab_{macro-index} is the actual data object which holds the data. As described above, this is a 1-D array; its size is *pf-data-size*, an integer variable set by the macros *pf-ragged-alloc* and *pf-ragged-realloc*.

[/Users/dstotler/degas2/src/pmiformat.hweb] Routines using PMI data.

pf-data-table(*i, j, k, l, m, n*) Multidimensional accessor function to the PMI data provided for ease of use; the integers *i* through *m* are the independent variable indices, $0 \leq i \leq pf_tab_index_{n,1}-1$, etc. where *n* is the dependent variable.

pf-ragged-alloc(*pf-data, pf-tab-index*) Computes sizes and base addresses of the one-dimensional data array *pf-data-tab* (*pf-data* is a root name here) based on the multi-dimensional information provided by *pf-tab-index*. There is additional documentation of a similar macro in *pmiformat.hweb*.

pf-ragged-realloc(*pf-data, pf-tab-index*) Makes changes to size of one-dimensional data array *pf-data-tab* based on a revised *pf-tab-index*. The user must be adding to the end of the table for this to work properly.

43 Problem definitions

[/Users/dstotler/degas2/src/problem.hweb]

\$Id: c74d310001403a259636d281a3c8926f5eeeca03d \$

[/Users/dstotler/degas2/src/problem.hweb] Information about problem particles is held in arrays in common blocks. An problem is identified by an integer indexing into these arrays.

```
"classes.f" 43.1 ≡
@f pr_test_decl integer
@f pr_background_decl integer
@f pr_reaction_decl integer
@f pr_generic_decl integer
@f pr_materials_decl integer
@f pr_pmi_decl integer

@m pr_test_args(x) x
@m pr_background_args(x) x
@m pr_reaction_args(x) x
@m pr_generic_args(x) x
@m pr_materials_args(x) x
@m pr_pmi_args(x) x

@m pr_test_dummy(x) x
@m pr_background_dummy(x) x
@m pr_reaction_dummy(x) x
@m pr_generic_dummy(x) x
@m pr_materials_dummy(x) x
@m pr_pmi_dummy(x) x

@m pr_test_decl(x) integer x
@m pr_background_decl(x) integer x
@m pr_reaction_decl(x) integer x
@m pr_generic_decl(x) integer x
@m pr_materials_decl(x) integer x
@m pr_pmi_decl(x) integer x

@m pr_test_copy(x,y) y = x
@m pr_background_copy(x,y) y = x
@m pr_reaction_copy(x,y) y = x
@m pr_generic_copy(x,y) y = x
@m pr_materials_copy(x,y) y = x
@m pr_pmi_copy(x,y) y = x

@m pr_test_check(x) (x > 0 ∧ x ≤ pr_test_num)
@m pr_background_check(x) (x > 0 ∧ x ≤ pr_background_num)
@m pr_ex_test_check(x) (x > 0 ∧ x ≤ pr_ex_test_num)
@m pr_reaction_check(x) (x > 0 ∧ x ≤ pr_reaction_num)
@m pr_materials_check(x) (x > 0 ∧ x ≤ pr_materials_num)
@m pr_pmi_check(x) (x > 0 ∧ x ≤ pr_pmi_num)

@m pr_test(x) problem_test_sp_x
@m pr_background(x) problem_background_sp_x
@m pr_ex_test(x) problem_ex_test_sp_x
@m pr_reaction(x) problem_rc_x
@m pr_mat_ref(x) problem_materials_ref_x

@m pr_rc_num(x) problem_reaction_num_x
@m pr_ts_rc(x,i) problem_test_reactioni,x
@m pr_ts_bk(x,i) problem_test_backgroundi,x

@m pr_num_arrangements(i,j) problem_num_arrangementsj,i
```

```

@m pr_ts_prod(i,j,k,l) problem_test_productsl, k, j, i
@m pr_prod_mult(i,j,k) problem_prod_multk, j, i

@m pr_bk_rc(i) problem_background_reactioni
@m pr_bkrc_rg(i,j) problem_bkrc_reagentsj, i
@m pr_bkrc_prod(i,j) problem_bkrc_productsj, i

@m pr_ex_rc(i) problem_external_reactioni
@m pr_ex_ts_rc_num(i) problem_external_test_reaction_numi
@m pr_ex_ts_rc(x,i) problem_external_test_reactioni, x
@m pr_ex_ts_bk(x,i) problem_external_test_backgroundi, x
@m pr_ex_rc_prod(i,j,k) problem_exrc_productsk, j, i

@m pr_pm_ref(x) problem_pmi_refx
@m pr_pm_case_num(x) problem_pmi_case_numx
@m pr_pm_cases(x,i) problem_pmi_casesi, x

@m pr_pm_num_arrange(i,j) problem_pmi_num_arrangej, i
@m pr_pm_prod(i,j,k,l) problem_pmi_productsl, k, j, i
@m pr_pm_prod_mult(i,j,k) problem_pmi_prod_multk, j, i

@m pr_test_lookup(sp) problem_species_testsp
@m pr_background_lookup(sp) problem_species_backgroundsp
@m pr_reaction_lookup(rc) int_lookup(rc, problem_rc, pr_reaction_num)
@m pr_ex_test_lookup(sp) int_lookup(sp, problem_ex_test_sp, pr_ex_test_dim)

```

[/Users/dstotler/degas2/src/problem.hweb] Length specifications.

```
"classes.f" 43.2 ≡
@m pr_reaction_max 15
@m pr_arrangement_max 4
@m pr_pmi_max 8
@m pr_max_equiv 11
@m pr_bkrc_reagent_max 2
@m pr_tag_string_length 40
@m pr_max_lines 6
```

[/Users/dstotler/degas2/src/problem.hweb] Pointers to entries in dependent variable list. These are associated with string descriptors in subroutine *init_var0_list*. New additions here must be added there as well.

```
"classes.f" 43.3 ≡
@m pr_var_unknown 1
@m pr_var_mass 2 // test particle data
@m pr_var_momentum_vector 3
@m pr_var_momentum_2 4
@m pr_var_momentum_3 5
@m pr_var_energy 6
@m pr_var_angle 7
@m pr_var_emitter_v_vector 8 // For spectrum construction
@m pr_var_emitter_v_2 9
@m pr_var_emitter_v_3 10
@m pr_var_emitter_vf_Maxwell_vector 11
@m pr_var_emitter_vf_Maxwell_2 12
@m pr_var_emitter_vf_Maxwell_3 13
@m pr_var_emitter_vth_Maxwell 14
@m pr_var_xy_stress 15 // For Couette flow example

@m pr_num_diag_vars 5 // Constructed variables begin after
// these; place other constants above.
@m pr_var_mass_change 20
@m pr_var_momentum_change_vector 21 // Exchanges with background species
@m pr_var_momentum_change_2 22
@m pr_var_momentum_change_3 23
@m pr_var_energy_change 24
@m pr_var_mass_in 25 // Direction dependent for use with
@m pr_var_momentum_in_vector 26 // PMI and diagnostic sectors.
@m pr_var_momentum_in_2 27
@m pr_var_momentum_in_3 28 // "mass" variables are assumed to be
@m pr_var_energy_in 29 // first in these two lists. The lists
@m pr_var_mass_out 30 // are pr_num_diag_vars items long.
@m pr_var_momentum_out_vector 31
@m pr_var_momentum_out_2 32
@m pr_var_momentum_out_3 33
@m pr_var_energy_out 34

@m pr_num_change_vars 15

@if 0
@m pr_var_back_index(i, back) i + pr_num_change_vars * back
#endif

@m pr_problem_sp_back(back) back
@m pr_problem_sp_test(test) (pr_background_num + test)
@m pr_var_problem_sp_index(i, psp) i + pr_num_change_vars * psp
```

```
[/Users/dstotler/degas2/src/problem.hweb] Variable definitions.

"classes.f" 43.4 ≡
  package_init(pr)
  define_var_pk(pr, pr_test_num, INT)
  define_var_pk(pr, pr_background_num, INT)
  define_var_pk(pr, pr_ex_test_num, INT)
  define_var_pk(pr, pr_ex_test_dim, INT)
  define_var_pk(pr, pr_reaction_num, INT)
  define_var_pk(pr, pr_reaction_dim, INT)
  define_var_pk(pr, pr_bkrc_num, INT)
  define_var_pk(pr, pr_bkrc_dim, INT)
  define_var_pk(pr, pr_exrc_num, INT)
  define_var_pk(pr, pr_exrc_dim, INT)
  define_var_pk(pr, pr_materials_num, INT)
  define_var_pk(pr, pr_pmi_num, INT)
  define_var_pk(pr, pr_var0_num, INT)
  define_dimen_pk(pr, problem_test_ind, pr_test_num)
  define_dimen_pk(pr, problem_background_ind, pr_background_num)
  define_dimen_pk(pr, problem_ex_test_ind, pr_ex_test_dim)
  define_dimen_pk(pr, problem_reaction_ind, pr_reaction_dim)
  define_dimen_pk(pr, problem_reaction_ind0, pr_reaction_max)
  define_dimen_pk(pr, problem_bkrc_ind, pr_bkrc_dim)
  define_dimen_pk(pr, problem_bkrc_rg_ind, pr_bkrc_reagent_max)
  define_dimen_pk(pr, problem_exrc_ind, pr_exrc_dim)
  define_dimen_pk(pr, problem_species_ind, sp_num)
  define_dimen_pk(pr, problem_materials_ref_ind, ma_num)
  define_dimen_pk(pr, problem_materials_sub_ind, pr_materials_num)
  define_dimen_pk(pr, problem_pmi_ref_ind, pm_num)
  define_dimen_pk(pr, problem_pmi_sub_ind, pr_pmi_num)
  define_dimen_pk(pr, problem_pmi_ind0, pr_pmi_max)
  define_dimen_pk(pr, problem_product_ind, rc_product_max)
  define_dimen_pk(pr, problem_arr_ind, pr_arrangement_max)
  define_dimen_pk(pr, pr_var0_list_ind, pr_var0_num)
  define_dimen_pk(pr, pr_tag_string, pr_tag_string_length)
  define_varp_pk(pr, problem_species_test, INT, problem_species_ind)
  define_varp_pk(pr, problem_species_background, INT, problem_species_ind)
  define_varp_pk(pr, problem_materials_sub, INT, problem_materials_ref_ind)
  define_varp_pk(pr, problem_test_sp, INT, problem_test_ind)
  define_varp_pk(pr, problem_background_sp, INT, problem_background_ind)
  define_varp_pk(pr, problem_ex_test_sp, INT, problem_ex_test_ind)
  define_varp_pk(pr, problem_materials_ref, INT, problem_materials_sub_ind)
  define_varp_pk(pr, problem_rc, INT, problem_reaction_ind)
  define_varp_pk(pr, problem_reaction_num, INT, problem_test_ind)
  define_varp_pk(pr, problem_test_reaction, INT, problem_reaction_ind0, problem_test_ind)
  define_varp_pk(pr, problem_test_background, INT, problem_reaction_ind0, problem_test_ind)
  define_varp_pk(pr, problem_num_arrangements, INT, problem_reaction_ind0, problem_test_ind)
  define_varp_pk(pr, problem_test_products, INT, problem_product_ind, problem_arr_ind,
                 problem_reaction_ind0, problem_test_ind)
  define_varp_pk(pr, problem_prod_mult, FLOAT, problem_arr_ind, problem_reaction_ind0,
                 problem_test_ind)
```

```

define_varp_pk(pr, problem_background_reaction, INT, problem_bkrc_ind)
define_varp_pk(pr, problem_bkrc_reagents, INT, problem_bkrc_rg.ind, problem_bkrc_ind)
define_varp_pk(pr, problem_bkrc_products, INT, problem_product_ind, problem_bkrc_ind)

define_varp_pk(pr, problem_external_reaction, INT, problem_exrc.ind)
define_varp_pk(pr, problem_external_test_reaction_num, INT, problem_ex_test.ind)
define_varp_pk(pr, problem_external_test_reaction, INT, problem_reaction.ind0, problem_ex_test.ind)
define_varp_pk(pr, problem_external_test.background, INT, problem_reaction.ind0, problem_ex_test.ind)
define_varp_pk(pr, problem_exrc_products, INT, problem_product.ind, problem_reaction.ind0,
               problem_ex_test.ind)

define_varp_pk(pr, problem_pmi_ref, INT, problem_pmi_sub.ind)
define_varp_pk(pr, problem_pmi_sub, INT, problem_pmi_ref.ind)
define_varp_pk(pr, problem_pmi_case_num, INT, problem_test.ind)
define_varp_pk(pr, problem_pmi_cases, INT, problem_pmi.ind0, problem_test.ind)

define_varp_pk(pr, problem_pmi_num_arrange, INT, problem_pmi.ind0, problem_test.ind)
define_varp_pk(pr, problem_pmi_products, INT, problem_product.ind, problem_arr.ind,
               problem_pmi.ind0, problem_test.ind)
define_varp_pk(pr, problem_pmi_prod_mult, FLOAT, problem_arr.ind, problem_pmi.ind0,
               problem_test.ind)

define_varp_pk(pr, pr_var0_list, CHAR, pr_tag_string, pr_var0_list.ind)

define_varlocal_pk(pr, problem_version, CHAR, string)
package_end(pr)

```

[/Users/dstotler/degas2/src/problem.hweb] Variables for problemsetup.

```

"classes.f" 43.5 ≡
  package_init(ps)
  define_var_pk(ps, num_generics, INT)
  define_dimen_pk(ps, pr_equiv.ind, pr_max_equiv)
  define_varp_pk(ps, generics, INT, problem_species.ind)
  define_varp_pk(ps, num_equiv, INT, problem_species.ind)
  define_varp_pk(ps, equivalents, INT, pr_equiv.ind, problem_species.ind)
  package_end(ps)

```

44 Problem class attribute descriptions

[/Users/dstotler/degas2/src/problem.hweb]

[`/Users/dstotler/degas2/src/problem.hweb`] Data structures used to specify the current problem. Prefix is *pr*.

These properties characterize the interactions between the various components of the problem: test particles, background species, reactions, materials, and plasma-material interactions. The integer indices and array values noted below correspond to entries into one of several lists. For clarity, we will associate with each list a different subscript to be used in the subsequent definitions:

1. Reference list of species (class *sp*) (subscript: *s*),
2. Reference list of reactions (class *rc*) (subscript: *r*),
3. List of test species (problem input) (subscript: *t*),
4. List of background species (problem input) (subscript: *b*),
5. List of problem reactions (problem input) (subscript: *pr*),
6. List of problem background reactions (problem input) (subscript: *br*),
7. Reference list of materials (class *ma*) (subscript: *m*),
8. List of problem materials (problem input) (subscript: *ms*),
9. Reference list of plasma-materials interactions (class *pm*) (subscript: *p*),
10. List of problem PMI (problem input) (subscript: *ps*),

The index *i* below refers to an element of each of these lists. Once we've got the subset lists specified in the "problem input", we can identify the reactions and PMI in which particular test species participates. Indices for these sub-subset lists are denoted by *j* below. Reactions between background species only (e.g., recombination) are designated as "background reactions" and are characterized by separate arrays below. Background reactions enter into the problem by having a test species (maybe more than one?) in their list of products and, thus, give rise to a source group (class *so*).

The third class of reactions, "external reactions", are not used directly in the DEGAS 2 calculation, but by a plasma code coupled to it. Associated with these reactions is a list of "external" test species that play a role analogous to standard test species. The "background" species in external reactions are sufficiently similar to standard background species that no distinction is made. *Both the external test and reaction specifications are completely optional and will not be present in most problem files.*

pr_test(i_t) Species index of test species *i_t*.

pr_background(i_b) Species index of background species *i_b*.

pr_ex_test(i_t) Species index of external test species *i_t*.

pr_reaction(i_{pr}) Index into the reference list of reactions for problem reaction *i_{pr}*.

pr_mat_ref(i_{ms}) Reference materials index of the *i_{ms}* entry in the problem materials subset.

pr_rc_num(i_t) Number of reactions in which test species *i_t* participates.

pr_ts_rc(i_t,j) Index of the problem reaction for the *j*th reaction involving test species *i_t*, $1 \leq j \leq pr.rc.num(i_t)$.

$pr_ts_bk(i_t, j)$ Index into the list of (problem) background species for the j th reaction involving test species i_t .

$pr_num_arrangements(i_t, j)$ Number of arrangements of products (which may or may not be truly different) for the j th reaction involving test species i_t .

$pr_ts_prod(i_t, j, k, l)$ The l th product (equivalent to the l th entry in the list of generic species given by $rc_product$) for the k th arrangement of the j th reaction involving test species i_t .

$pr_prod_mult(i_t, j, k)$ Relative weight or multiplicity of the k th arrangement of the j th reaction involving test species i_t . This quantity is the probability of obtaining arrangement k in this reaction.

$pr_bk_rc(i_{br})$ Index into the reference list of reactions for problem background reaction i_{br} .

$pr_bkrc_rg(i_{br}, j)$ Index into the list of (problem) background species for the j th reagent ($1 \leq j \leq pr_bkrc_reagent_max$) in the i_{br} th background reaction.

$pr_bkrc_prod(i_{br}, j)$ Index into the list of reference species for the j th product of the i_{br} th background reaction.

$pr_ex_rc(i_{ex})$ Index into the reference list of reactions for external reaction i_{ex} .

$pr_ex_ts_rc_num(i_t)$ Number of external reactions in which external test species i_t participates.

$pr_ex_ts_rc(i_t, j)$ Index of the problem background species for the j th reaction involving external test species i_t , $1 \leq j \leq pr_ex_rc_num(i_t)$.

$pr_ex_ts_bk(i_t, j)$ Index of the problem (external) reaction for the j th reaction involving external test species i_t .

$pr_ex_rc_prod(i_t, j, k)$ Reference species index for the k th product of the j th reaction involving external test species i_t .

$pr_pm_ref(i_{ps})$ Index into the reference list of PMI for problem PMI i_{ps} .

$pr_pm_case_num(i_t)$ Number of PMI in which test species i_t participates.

$pr_pm_cases(i_t, j)$ Index of the problem PMI for the j th reaction involving test species i_t , $1 \leq j \leq pr_pm_case_num(i_t)$.

$pr_pm_num_arrange(i_t, j)$ Number of arrangements of products (which may or may not be truly different) for the j th PMI involving test species i_t .

$pr_pm_prod(i_t, j, k, l)$ The l th product for the k th arrangement of the j th PMI involving test species i_t .

$pr_prod_mult(i_t, j, k)$ Relative weight or multiplicity of the k th arrangement of the j th PMI involving test species i_t . This quantity is the probability of obtaining arrangement k in this PMI.

[/Users/dstotler/degas2/src/problem.hweb] Internal variables for the problem class.

pr_test_num Number of test species in the problem.

pr_background_num Number of background species in the problem.

pr_ex_test_num Number of external test species in use.

pr_reaction_num Number of reactions in the problem.

pr_bkrc_num Number of background reactions in the problem.

pr_bkrc_dim For dimensioning; is equal to *pr_bkrc_num*, except is set to 1 when *pr_bkrc_num*=0.

pr_exrc_num Number of external reactions in use.

pr_exrc_dim For dimensioning; is equal to *pr_exrc_num*, except is set to 1 when *pr_exrc_num*=0.

pr_materials_num Number of materials in the problem.

pr_pmi_num Number of plasma-material interactions in the problem.

pr_var0_num Number of entries in *pr_var0_list*.

pr_materials_sub[*i_m*] Problem index of reference material *i_m*.

problem_pmi_sub[*i_p*] Index of the problem PMI corresponding to reference PMI *i_p*.

pr_var0_list[*j*] List of dependent variables in the problem (compiled on the fly), $1 \leq j \leq pr_var0_num$.

[/Users/dstotler/degas2/src/problem.hweb] Routines using problem class.

Note: these first two lookup macros are frequently used and are thus based on actual internal arrays.

pr_test_lookup(*i_s*) Index into the list of problem test species corresponding to reference species *i_s*; returns 0 if the species *i_s* is not a test species.

pr_background_lookup(*i_s*) Index into the list of problem background species corresponding to reference species *i_s*; returns 0 if the species *i_s* is not a background species.

pr_reaction_lookup(*i_r*) Index into the list of problem reactions corresponding to reference reaction *i_r*.

pr_problem_sp_back(*back*) Provides integer index into combined background and test list (problem species list) for background species *back*.

pr_problem_sp_test(*test*) Provides integer index into combined background and test list (problem species list) for test species *test*.

pr_var_problem_sp_index(*i,psp*) Provides integer index into *pr_var0_list* for any problem species *psp*. The list of problem species is the combined background and test species list. (The generic exchange variables such as *pr_var_mass_change* are duplicated *pr_background_num* + *pr_test_num* times).

[/Users/dstotler/degas2/src/problem.hweb] Problem setup variables for isotopic variations. Prefix is *ps*.

The following variables are required in a number of the *problemsetup.web* routines, but not in any outside; hence, they have been broken off into this separate package defined within *problem.hweb*.

The mechanism for dealing with isotopic variations involves defining a “generic species”, a single species which represents all of the isotopic combinations equivalent to it. E.g., for the all-important case of hydrogen, the generic archetype for a given molecule is defined (or at least it should be) as the one containing only protium isotopes.

num_generics Total number of “generic species” appearing in the lists of test and background species.

generics[*i_s*] Provides the index *i_g* (used below) of the “generic species” which is being used to represent (the isotopically equivalent) species *i_s* of the reference species list.

num_equiv[*i_g*] Total number of entries in the test and background species lists which are equivalent to “generic species” *i_g*.

equivalents[*i_g,j*] Provides the index into the reference species list of the *j*th particle equivalent to generic species *i_g*, where $1 \leq j \leq \text{num_equiv}[i_g]$.

45 Reaction data definitions

[/Users/dstotler/degas2/src/reactiondata.hweb]

\$Id: 6db05bfa698a39ee7f7ea2f201f19737c99bcd9 \$

[/Users/dstotler/degas2/src/reactiondata.hweb] Data for reactions is held in arrays in common blocks. *rd_rate*(*i, j, k, r*) accesses the reaction rate for the *r*th reaction as an array with up to 3 dimensions. “Handling” (i.e., not reaction rate) data is kept in the analogous *rd_handling*(*i, j, k, d, r*) array. The *d* index denotes the *d*th dependent variable for the *r*th reaction.

```
"classes.f" 45.1 ≡
@m rd_rate(i,j,k,r)
    reaction_rate_tabreaction_rate_base_r+i+reaction_rate_tab_index_{r,1}*(j+reaction_rate_tab_index_{r,2}*k)}

@m rd_handling(i,j,k,d,r)
    reaction_handling_tabreaction_handling_base_{r,d}+i+reaction_handling_tab_index_{r,d,1}*(j+reaction_handling_tab_index_{r,d,2}*k)}

@m rd_data_args(x,subs)
    x##_eval_name##subs, x##_rank##subs, x##_var##subs1, x##_tab_index##subs1,
    x##_spacing##subs0, x##_min##subs1, x##_delta##subs1, x##_tab_x##_base##subs

@m rd_dep_var_max $EVAL (xs_dep_var_max - 1)
```

```
[/Users/dstotler/degas2/src/reactiondata.hweb] Variable definitions.

"classes.f" 45.2 ≡
    package_init(rd)
        define_dimen_pk(rd, rd_rank_ind, xs_table_rank_max)
        define_dimen_pk(rd, rd_dep_var_ind, rd_dep_var_max)
        define_dimen_pk(rd, rd_rank_ind0, 0, xs_table_rank_max)
        define_dimen_pk(rd, rd_tag_string, xs_tag_string_length)
        define_dimen_pk(rd, rd_eval_string, xs_eval_name_length)

        define_var_pk(rd, reaction_rate_size, INT)
        define_dimen_pk(rd, rd_table_ind, 0, reaction_rate_size - 1)
        define_var_pk(rd, reaction_handling_size, INT)
        define_dimen_pk(rd, rd_handling_table_ind, 0, reaction_handling_size - 1)

        define_varp_pk(rd, reaction_rate_min, FLOAT, rd_rank_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_delta, FLOAT, rd_rank_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_rank, INT, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_spacing, INT, rd_rank_ind0, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_eval_name, CHAR, rd_eval_string, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_tab_index, INT, rd_rank_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_var, INT, rd_rank_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_num_rand, INT, problem_reaction_ind)

        define_varp_pk(rd, reaction_rate_base, INT, problem_reaction_ind)
        define_varp_pk(rd, reaction_rate_tab, FLOAT, rd_table_ind)

        define_varp_pk(rd, reaction_handling_min, FLOAT, rd_rank_ind, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_delta, FLOAT, rd_rank_ind, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_rank, INT, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_spacing, INT, rd_rank.ind0, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_eval_name, CHAR, rd_eval_string, rd_dep_var_ind,
                      problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_tab_index, INT, rd_rank_ind, rd_dep_var_ind,
                      problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_var0, INT, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_var, INT, rd_rank_ind, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_num_rand, INT, problem_reaction_ind)

        define_varp_pk(rd, reaction_handling_base, INT, rd_dep_var_ind, problem_reaction_ind)
        define_varp_pk(rd, reaction_handling_tab, FLOAT, rd_handling_table_ind)
    package_end(rd)
```

[/Users/dstotler/degas2/src/reactiondata.hweb] Common blocks definitions.

```
"classes.f" 45.3 ≡
@f rd_decls integer
@m rd_decls integer inc

/* The ragged_alloc macros are the same as those in pmidata.hweb. More detailed documentation
   is given there. */
@m rd_set_baseinc(x,y,subs) x##_base##subs## = x##_size
  inc = 1 $DO (I, range_min(ind_##y##_1), range_max(ind_##y##_1))
  {
    inc = inc * y##subs##_I
  }

@m rd_ragged_alloc(x) x##_size = 0
  var_alloc(x##_base)
  call init_base(x##_base, array_size(x##_base))

@m rd_ragged_realloc(x,y,subs) rd_set_baseinc(x, y, subs)
  var_realloc(x##_tab, x##_size - 1, x##_size - 1 + inc)
  x##_size = x##_size + inc

@m rd_null_alloc(x) var_realloc(x##_tab, x##_size - 1, x##_size - 1 + 1)
  x##_size = 1
```

46 Reaction data class attribute descriptions

[/Users/dstotler/degas2/src/reactiondata.hweb]

[`/Users/dstotler/degas2/src/reactiondata.hweb`] Structures ot hold atomic physics data for the current problem. Prefix is *rd*.

A given reaction within this class is identified by the integer index *pr_reac* (“problem reaction”) assigned to it during the problem setup phase. This class collates the “cross section” data (class *xs*) for all of the reactions in one place for the main code to use during the run. The reaction data consist of two very similar subclasses. The first has names which begin with *reaction_rate*; these arrays are used only to contain the reaction rate for a particular *pr_reac*. The second subclass has names starting with *reaction_handling* and is used for the various data needed to process or handle a particular collision, along with scoring. The nature and names of the data objects within the *reaction_handling* array can vary not only with the reaction, but with the model used to represent a given reaction.

The representation of the data in memory is crucial to efficiency. Since the main data tables here are four or five dimensional objects of very “ragged” structure, we have opted for an internal representation as single dimensional arrays. To simplify coding, macros have been developed to simulate the familiar multi-dimensional array functionality. The downside of this is the addition of bookkeeping arrays (here, the *xxx_base* arrays), the unwieldy-ness of the macros themselves, and the difficulty in extracting data from the 1-D array during debugging or while perusing the netCDF files. At least this last issue may be addressed by the construction of a small browser code to allow multi-dimensional netCDF data to be viewed and plotted in a straightforward manner. The macros used in *pmidata.hweb* are very similar to these; an attempt at documenting their operation can be found in that file.

reaction_rate_size Size of the 1-D array for the reaction rates; set by the macros *rd_ragged_alloc* and *rd_ragged_realloc*.

reaction_handling_size Size of the 1-D array for the handling data; set by the macros *rd_ragged_alloc* and *rd_ragged_realloc*.

reaction_rate_min_{pr_reac,i} Floating point minimum value of the *i*th independent variable of the rate for *pr_reac*.

reaction_rate_delta_{pr_reac,i} Floating point increment in the *i*th independent variable of the rate for *pr_reac*.

reaction_rate_rank_{pr_reac} Integer number of independent variables in the rate for *pr_reac*.

reaction_rate_spacing_{pr_reac,i} Integer identifying the way the *i*th variable of the rate (*i* = 0 refers to the rate itself) are to be interpolated. Examples are *xs_spacing_linear* and *xs_spacing_log*. Namely, in the latter case the log of the independent variable (or dependent if *i* = 0) is interpolated linearly.

reaction_rate_eval_name_{pr_reac} String providing the name of the function which will be used to evaluate the rate of *pr_reac*. The only currently implemented option is “table”; more generally, the name of a fitting function would be possible.

reaction_rate_tab_index_{pr_reac,i} Number of values used for the *i*th independent variable of the rate for *pr_reac*.

reaction_rate_var_{pr_reac,i} Integer identifying the *i*th independent variable for the rate. The independent variables are typically temperature, density, or energy; see *xsection.hweb* for a full list.

reaction_rate_num_rand_{pr_reac} Integer number of uniform random deviates required to evaluate the rate for *pr_reac* (will usually be 0 here, however).

reaction_rate_base_{pr_reac} Index in *reaction_rate_tab* at which the data for the rate of *pr_reac* begins. This array is set by the macros *rd_ragged_alloc* and *rd_ragged_realloc*.

reaction_rate_tab_{macro_index} is the actual data object which holds the reaction rate data. As described above, this is a 1-D array of size *reaction_rate_size*. These data are normally accessed through the macro interface *rd_rate* (see below).

reaction_handling_min_{pr_reac,j,i} Floating point minimum value of the *i*th independent variable for the *j*th dependent variable in *pr_reac*.

reaction_handling_delta_{pr_reac,j,i} Floating point increment in the *i*th independent variable for the *j*th dependent variable in *pr_reac*.

reaction_handling_rank_{pr_reac,j} Integer number of independent variables for the *j*th dependent variable in *pr_reac*.

reaction_handling_spacing_{pr_reac,j,i} Integer identifying the way the *i*th variable for the *j*th dependent variable (*i* = 0 refers to the dependent variable itself) is to be interpolated. Examples are *xs_spacing_linear* and *xs_spacing_log*. Namely, in the latter case the log of the independent variable (or dependent if *i* = 0) is interpolated linearly.

reaction_handling_eval_name_{pr_reac,j} String providing the name of the function which will be used to evaluate the data for the *j*th dependent variable of *pr_reac*. The only currently implemented option is “table”; more generally, the name of a fitting function would be possible.

reaction_handling_tab_index_{pr_reac,j,i} Number of values used for the *i*th independent variable of the *j*th dependent variable for *pr_reac*.

reaction_handling_var0_{pr_reac,j} Integer identifying the *j*th dependent variable of *pr_reac*. This value is a pointer into the global dependent variable list *pr_var0_list*.

reaction_handling_var_{pr_reac,j,i} Integer identifying the *i*th independent variable for the *j*th dependent variable of *pr_reac*. The independent variables are typically temperature, density, or energy; see *xsection.hweb* for a full list.

reaction_handling_num_rand_{pr_reac} Integer number of uniform random deviates required to evaluate all dependent variables of *pr_reac*. For example, the integral of a distribution may be broken into a number of equal pieces so that it can be sampled by interpolation with a single random number. Since the same random number may be needed for more than one dependent variable (e.g., coupled, multi-dimensional distributions), this variable contains only the total number of random deviates.

reaction_handling_base_{pr_reac,j} Index in *reaction_handling_tab* at which the data for the *j*th dependent variable of *pr_reac* begin. This array is set by the macros *rd_ragged_alloc* and *rd_ragged_realloc*.

reaction_handling_tab_{macro_index} Is the actual data object which holds the handling data. As described above, this is a 1-D array of size *reaction_handling_size*. These data are normally accessed through the macro interface *rd_handling* (see below).

[/Users/dstotler/degas2/src/reactiondata.hweb] Routines using reaction data.

rd_rate(i, j, k, pr_reac) Multidimensional accessor function to the reaction rate data provided for easier use; i, j, k are the independent variable indices, $0 \leq i \leq xs_tab_index_{pr_reac,1}-1$, etc. for the reaction *pr_reac*.

rd_handling(i, j, k, d, pr_reac) Multidimensional accessor function to the handling data provided for easier use; i, j, k are the independent variable indices, $0 \leq i \leq xs_tab_index_{pr_reac,d,1}-1$, etc. for the *d*th dependent variable of reaction *pr_reac*.

rd_data_args(x, subs) This macro provides an argument list of all of the *reaction_rate* or *reaction_handling* (use either of these as *x*) arrays for passing down to a subroutine. The *subs* argument provides the *pr_reac* array index (both cases) and the dependent variable index for *reaction_handling*. E.g., usage might be: *rd_data_args(reaction_rate, pr_reac)* or *rd_data_args(reaction_handling, pr_reac,d)*.

rd_ragged_alloc(rd_data) Initializes *rd_data_size* and allocates space for *rd_data_base* (*rd_data* is a root name here, either *reaction_rate* or *reaction_handling*). This macro needs to do nothing more because the reaction data are loaded in one reaction at a time, and the sizes are not known a priori.

rd_ragged_realloc(rd_data, rd_tab_index, subs) Makes changes to size of one-dimensional data array *rd_data_tab* (*rd_data* is a root name here, either *reaction_rate* or *reaction_handling*) based on a revised *rd_tab_index*. *subs* provides the *pr_reac* array index (both cases) and the dependent variable index for *reaction_handling*. The user must be adding to the end of the table for this to work properly.

47 Plasma-Material Interaction (PMI) data definitions

[/Users/dstotler/degas2/src/pmidata.hweb]

\$Id: cc2a235d694e0640f0498ac5a717424f9edf1be1 \$

[/Users/dstotler/degas2/src/pmidata.hweb] Data for Plasma Material Interactions is held in arrays in common blocks. *pd_yield(i, j, k, l, m, p)* accesses the data for the *p*th PMI as an array with up to 5 dimensions. Data used in handling the PMI are stored in the *pd_handling(i, j, k, l, m, d, p)* array with the *d* subscript denoting the *d*th dependent variable associated with the *p*th PMI.

```
"classes.f" 47.1 ≡
@M pd_yield(i,j,k,l,m,p)
  pmi_yield_tabi+pmi_yield_tab_indexp,1*(j+pmi_yield_tab_indexp,2*(k+pmi_yield_tab_indexp,3*(l+pmi_yield_tab_indexp,4*m)))+pmi_yiel
```

```
@M pd_handling(i,j,k,l,m,d,p)
  pmi_handling_tabi+pmi_handling_tab_indexp,d,1*(j+pmi_handling_tab_indexp,d,2*(k+pmi_handling_tab_indexp,d,3*(l+pmi_handling_ta
```

```
@M pd_data_args(x,subs)
  x##_eval_name##subs, x##_rank##subs, x##_var##subs1, x##_tab_index##subs1,
  x##_spacing##subs0, x##_min##subs1, x##_delta##subs1, x##_tab0, x##_base##subs
```

```
@M pd_dep_var_max $EVAL (pf_dep_var_max - 1)
```

[/Users/dstotler/degas2/src/pmidata.hweb] Variable definitions.

```
"classes.f" 47.2 ≡
  package_init(pd)
    define_dimen_pk(pd, pd_rank_ind, pf_table_rank_max)
    define_dimen_pk(pd, pd_dep_var_ind, pd_dep_var_max)
    define_dimen_pk(pd, pd_rank_ind0, 0, pf_table_rank_max)
    define_dimen_pk(pd, pd_tag_string, pf_tag_string_length)
    define_dimen_pk(pd, pd_eval_string, pf_eval_string_length)

    define_var_pk(pd, pmi_yield_size, INT)
    define_dimen_pk(pd, pd_yield_ind, 0, pmi_yield_size - 1)
    define_var_pk(pd, pmi_handling_size, INT)
    define_dimen_pk(pd, pd_handling_ind, 0, pmi_handling_size - 1)

    define_varp_pk(pd, pmi_yield_min, FLOAT, pd_rank_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_delta, FLOAT, pd_rank_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_rank, INT, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_spacing, INT, pd_rank_ind0, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_eval_name, CHAR, pd_eval_string, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_tab_index, INT, pd_rank_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_var, INT, pd_rank_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_num_rand, INT, problem_pmi_sub_ind)

    define_varp_pk(pd, pmi_yield_base, INT, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_yield_tab, FLOAT, pd_yield_ind)

    define_varp_pk(pd, pmi_handling_min, FLOAT, pd_rank_ind, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_delta, FLOAT, pd_rank_ind, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_rank, INT, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_spacing, INT, pd_rank_ind0, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_eval_name, CHAR, pd_eval_string, pd_dep_var_ind,
                  problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_tab_index, INT, pd_rank_ind, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_var0, CHAR, pd_tag_string, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_var, INT, pd_rank_ind, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_num_rand, INT, problem_pmi_sub_ind)

    define_varp_pk(pd, pmi_handling_base, INT, pd_dep_var_ind, problem_pmi_sub_ind)
    define_varp_pk(pd, pmi_handling_tab, FLOAT, pd_handling_ind)
    define_varlocal_pk(pd, pd_inc, INT)
  package_end(pd)
```

[/Users/dstotler/degas2/src/pmidata.hweb] Common blocks definitions.

/* *pd_ragged_alloc* and *pd_ragged_realloc* are designed to automatically allocate memory for ragged data arrays. To save space, these arrays are packed into 1-D arrays and the multidimensionality is simulated through the macros at the top of this class. Two variables are used along with the main data table in this process. All three should be formed from the same “root” word or phrase; this simplifies the macro argument lists so that only *x* needs to be passed in. The *y* array provides the actual dimensions of the ragged array.

x_tab One-dimensional array which will contain the data.

x_size Integer variable representing the total size of *x_tab*.

x_base Multidimensional array which provides the “bookmarks” needed to separate the portions of *x_tab*.

The input variable *subs* represents the dependence on variables which the macro doesn’t need to know about (e.g., dependent variable index or reaction number).

y Dependence on indices external to this macros is again carried through the variable *subs*. The “rank” index used in the macros gives the size of the data required for this dependent variable in *x_tab*. The number of nontrivial ($\neq 1$; can’t use 0 as a trivial size because of the need to take products) entries in *y* gives the real dimensionality of this dependent variable. The value of *y* at each of these entries is the number of independent variables for that dimension.

x_base just keeps track of where each dependent variable begins in *x_tab*. The macro used to simulate the multidimensionality of *x_tab* uses both *x_base* and *y* to locate a particular entry. The size required for a dependent variable is just the product of its dimensions which can be computed (*pd_set_baseinc*) by looping over its rank and forming the product of the corresponding entries in *y*.

Since *x_tab* has zero size at the beginning of the code, no initial allocation is required. However, *x_size* must be initialized and memory for *x_base* allocated. We do this in the guise of *pd_ragged_alloc* for consistency with other macros which are expanded at the same time. */

```
"classes.f" 47.3 ≡
@m pd_set_baseinc(x,y,subs) x##_base##subs = x##_size
    pd_inc = 1 $DO (I, range_min(ind_##y##_1), range_max(ind_##y##_1))
    {
        pd_inc = pd_inc * y##subs_I
    }

@m pd_ragged_alloc(x) x##_size = 0
    var_alloc(x##_base)
    call init_base(x##_base, array_size(x##_base))

@m pd_ragged_realloc(x,y,subs) pd_set_baseinc(x, y, subs)
    var_realloc(x##_tab, x##_size - 1, x##_size - 1 + pd_inc)
    x##_size = x##_size + pd_inc

/* Although the end result of these macros is similar to that from those in pmiformat.hweb, the
   approach appears quite different. The reason is that in this case, none of the crucial dimensions
   are known at compile time. Instead, the base and increments are recomputed, and memory
   reallocated, not only as new data sets are read in, but also as each dependent variable is
   handled. */
```

48 PMI data class attribute descriptions

[/Users/dstotler/degas2/src/pmidata.hweb]

[`/Users/dstotler/degas2/src/pmidata.hweb`] Structures used to hold plasma-material interaction data for the current problem. Prefix is *pd*.

A given PMI is identified by an integer index *pr_pmi* (“problem PMI”). This is the internal class corresponding to the “PMI format” class *pf*. It contains all of the data required to deal with the PMI in the problem. Not by accident, the elements of this class closely resemble those of the reaction data class *rd*. One difference is that fits have been made available for some of the PMI.

For clarity, here are the assumptions made about the data used to represent the fit and the associated information in this class:

1. *rank* provides the number of independent variables required to evaluate the fit; their identity is stored in the corresponding *var* array just as for tables.
2. The data table passed (the fit coefficients) is considered to be strictly 1-D with *tab_index*₁ elements.
3. The multiplier *mult* does get applied to the fit coefficients. Hence, *mult*₀ should probably be set to one.
4. No spacing or value information is stored for the independent variables. Since this information is at least likely to be peculiar to each fit and at most irrelevant, it is assumed to reside within the fit function.
5. Eventually, we would like to store the fit function along with the data so that the data could be completely described by a single package, as can now be done with tables.

The PMI data class is divided into two similar subclasses. The first represents a single dependent variable, the “yield” of the PMI (e.g., the reflection coefficient, adsorption rate, sputtering yield, etc.; in effect, the probability of the process taking place). Hence, this sub-class lacks any explicit “dependent variable” dimension in the arrays. The *pmi_handling* arrays contain the data required to process or “handle” *pr_pmi*. The nature and names of the data objects within the *pmi_handling* array can vary not only with the PMI, but with the model used to represent a given reaction.

The representation of the data in memory is crucial to efficiency. Since the main data tables here are six or seven dimensional objects of very “ragged” structure, we have opted for an internal representation as single dimensional arrays. To simplify coding, macros have been developed to simulate the familiar multi-dimensional array functionality. The downside of this is the addition of bookkeeping arrays (here, the *xxx_base* arrays), the unwieldy-ness of the macros themselves, and the difficulty in extracting data from the 1-D array during debugging or while perusing the netCDF files. At least this last issue may be addressed by the construction of a small browser code to allow multi-dimensional netCDF data to be viewed and plotted in a straightforward manner. The macros are documented in more detail in *pmidata.hweb*.

pmi_yield_size Size of the 1-D array for the PMI yields; set by the macros *pd_ragged_alloc* and *pd_ragged_realloc*.

pmi_handling_size Size of the 1-D array for the handling data; set by the macros *pd_ragged_alloc* and *pd_ragged_realloc*.

*pmi_yield_min*_{*pr_pmi,i*} Floating point minimum value of the *i*th independent variable of the yield for *pr_pmi*.

*pmi_yield_delta*_{*pr_pmi,i*} Floating point increment in the *i*th independent variable of the yield for *pr_pmi*.

*pmi_yield_rank*_{*pr_pmi*} Integer number of independent variables for the yield for *pr_pmi*.

pmi_yield_spacing_{pr_pmi,i} Integer identifying the way the *i*th variable for the yield of *pr_pmi* (*i* = 0 refers to the yield itself) is to be interpolated. Examples are *pf_spacing_linear* and *pf_spacing_log*. Namely, in the log case the log of the independent variable (or dependent if *i* = 0) is interpolated linearly.

pmi_yield_eval_name_{pr_pmi} String (length *pf_eval_string_length*) providing the name of the function which will be used to evaluate the yield data. The most prevalent value is “table”; other values must correspond to the name of a fitting subroutine. See *plate.web* for more details.

pmi_yield_tab_index_{pr_pmi,i} Number of values used for the *i*th independent variable of the yield for *pr_pmi*.

pmi_yield_var_{pr_pmi,i} Integer identifying the *i*th independent variable for the yield. The independent variables are typically incident energy, and incident poloidal angle; see *pmiformat.hweb* for a full list.

pmi_yield_num_rand_{pr_pmi} Integer number of uniform random variates required to evaluate the yield for *pr_pmi*. This is provided for compatibility with the *pmi_handling* variables and will usually be 0 here.

pmi_yield_base_{pr_pmi} Index in *pmi_yield_tab* at which the data for the yield of *pr_pmi* begin. This array is set by the macros *pd_ragged_alloc* and *pd_ragged_realloc*.

pmi_yield_tab_{macro_index} Is the actual data object which holds the yield data. As described above, this is a 1-D array of size *pmi_yield_size*. These data are normally accessed through the macro interface *pd_yield*.

pmi_handling_min_{pr_pmi,j,i} Floating point minimum value of the *i*th independent variable for the *j*th dependent variable in *pr_pmi*.

pmi_handling_delta_{pr_pmi,j,i} Floating point increment in the *i*th independent variable for the *j*th dependent variable in *pr_pmi*.

pmi_handling_rank_{pr_pmi,j} Integer number of independent variables for the *j*th dependent variable in *pr_pmi*.

pmi_handling_spacing_{pr_pmi,j,i} Integer identifying the way the *i*th variable for the *j*th dependent variable (*i* = 0 refers to the dependent variable itself) is to be interpolated. Examples are *pf_spacing_linear* and *pf_spacing_log*. Namely, in the log case the log of the independent variable (or dependent if *i* = 0) is interpolated linearly.

pmi_handling_eval_name_{pr_pmi,j} String (length *pf_eval_string_length*) providing the name of the function or subroutine which will be used to evaluate the data for the *j*th dependent variable in handling *pr_pmi*. The most prevalent value is “table”; other values must correspond to the name of a fitting subroutine. See *plate.web* for more details. Note that in some instances, the appropriate dependent “variable” is really a vector (i.e., velocity); in this case, tabular evaluation (which can return only a single scalar) makes no sense.

pmi_handling_tab_index_{pr_pmi,j,i} Number of values used for the *i*th independent variable of the *j*th dependent variable for *pr_pmi*.

pmi_handling_var0_{pr_pmi,j} String (length *pf_tag_string_length*) identifying the *j*th dependent variable of *pr_pmi*. A string is used here since the nature of the handling data varies from one PMI to the next.

pmi_handling_var_{pr_pmi,j,i} Integer identifying the *i*th independent variable for the *j*th dependent variable. The independent variables are typically incident energy, and incident poloidal angle; see *pmiformat.hweb* for a full list.

pmi_handling_num_rand_{pr_pmi} Integer number of uniform random deviates required to handle a single interaction of *pr_pmi*. The need for this became apparent when implementing the Bateman format. In this case, the *same random number* is needed to evaluate more than one dependent variable. *pmi_handling_num_rand* lets the code maintain them through all of the required dependent variable evaluations.

pmi_handling_base_{pr_pmi,j} Index in *pmi_handling_tab* at which the data for the *j*th dependent variable of *pr_pmi* begin. This array is set by the macros *pd_ragged_alloc* and *pd_ragged_realloc*.

pmi_handling_tab_{macro_index} Is the actual data object which holds the handling data. As described above, this is a 1-D array of size *pmi_handling_size*. These data are normally accessed through the macro interface *pd_handling*.

[/Users/dstotler/degas2/src/pmidata.hweb] Routines using PMI data.

pd_yield(k, l, m, n, o, pr_pmi) The programmer's representation of the yield data for *pr_pmi*. *k, l, m, n*, and *o* represent the up-to-five dimensional dependency of the yield on its independent variables. *k* ranges from 0 → *pmi_yield_tab_index_{pr_pmi,1}* - 1; similarly for *l* with *i*=2, *m*, *i*=3, etc.

pd_handling(k, l, m, n, o, j, pr_pmi) The programmer's representation of the handling data for *pr_pmi*. *k, l, m, n*, and *o* represent the up-to-five dimensional dependency of the *j*th dependent variable on its independent variables. *k* ranges from 0 → *pmi_handling_tab_index_{pr_pmi,j,1}* - 1; similarly for *l* with *i*=2, *m*, *i*=3, etc.

pd_data_args(x, subs) This macro provides an argument list of all of the *pmi_yield* or *pmi_handling* (use either of these as *x*) arrays for passing down to a subroutine. The *subs* argument provides the *pr_pmi* array index (both cases) and the dependent variable index for *pmi_handling*. E.g., usage might be: *pd_data_args(pmi_yield, pr_pmi)* or *pd_data_args(pmi_handling, pr_pmi,j)*.

pd_ragged_alloc(pd_data) Initializes *pd_data_size* and allocates space for *pd_data_base* (*pd_data* is a root name here, either *pmi_yield* or *pmi_handling*). This macro needs to do nothing more because the PMI data are loaded in one PMI at a time, and the sizes are not known a priori.

pd_ragged_realloc(pd_data, pd_tab_index, subs) Makes changes to size of one-dimensional data array *pd_data_tab* (*pd_data* is a root name here, either *pmi_yield* or *pmi_handling*) based on a revised *pd.tab_index*. *subs* provides the *pr_pmi* array index (both cases) and the dependent variable index for *pmi_handling*. The user must be adding to the end of the table for this to work properly.

49 String definitions

[/Users/dstotler/degas2/src/string.hweb]

\$Id: f0e83637473f1fb5d226413f880dc504b276b8aa \$

Declarations of the string routines.

```
"classes.f" 49 ≡
@f st_decls integer
@m st_decls external string_length, read_string, parse_string, next_token, string_lookup, int_lookup,
read_real, read_integer, read_int_soft_fail, read_real_soft_fail, read_real_scaled
integer string_length, parse_string, string_lookup, int_lookup, read_integer, read_int_soft_fail
real read_real, read_real_soft_fail, read_real_scaled
logical read_string, next_token
```

[/Users/dstotler/degas2/src/string.hweb] A string without its trailing blanks

```
"classes.f" 49.1 ≡
@m SP $UNQUOTE('„')
@m trim(s) sSP(1 : string_length(s))
```

[/Users/dstotler/degas2/src/string.hweb] Read text from a string (does nothing)

```
"classes.f" 49.2 ≡
@m read_text(x) x
```

50 Macros for defining and manipulating arrays

[/Users/dstotler/degas2/src/array.hweb]

[/Users/dstotler/degas2/src/array.hweb] Macro definitions.

\$Id: cfaf9fb1a893ca6d19890e8dcc110613b4dd7c64 \$

```
"classes.f" 50.1 ≡
@m MAX_DIM 5
@m CHAR 0
@m INT 1
@m FLOAT 2
@m COMMA ,
@m paste(x, y) _paste(x, y)
@m _paste(x, y) x##y
```

[/Users/dstotler/degas2/src/array.hweb] Declaring dimensions and variables. Each vector that is declared has one or more dimensions which have names. These have to be defined first. Character variables have a name associated with their length. Each of these dimension names will have an upper and lower bound. @f **define_dimen integer** tells web to format the variables in **define_dimen** as integers. The macro **define_dimen** (*name* , *≠*) defines the lower and upper bounds of the dimension name *name*. It defines variables by prefixing min and max to the dimension name. If 1 arg. is given, it puts the lower limit (*min_name*) =1, and upper limit (*max_name*) = the first (and only) argument in **define_dimen**. If 2 args. are given, it puts the lower limit as the first arg and the upper limit as the second.

```
"classes.f" 50.2 ≡
@f define_dimen integer
/* #0 is the number of args represented by the dots.(atmost 2) ## pastes the strings on either side
   of it after appropriate expansions, hence name is expanded to the variable's name. #n represents
   the nth argument represented by the dots. */

@m define_dimen(name,...) $ASSERT (#0 ≡ 1 ∨ #0 ≡ 2)$IF (#0==_1,_$DEFINE(min_##name_1), \
$DEFINE(min_##name_#1))$IF (#0==_1,_$DEFINE(max_##name_1), \
$DEFINE(max_##name_#2))

/* Once the variables min_name and max_name are defined for each dimension name, the macro
   range(name) defines the range of the name. range_size(name) defines the size (upper limit -
   lower limit +1). */

@m range(name) range_min(name) : range_max(name)
@m rangea(name) range_min(name), range_max(name)
@m range_star(name) range_min(name) : *
@m range_stara(name) range_min(name) : range_min(name)
@m range_min1(name) min_##name
@m range_min(name) (range_min1(name))
@m range_max1(name) max_##name
@m range_max(name) (range_max1(name))
@m range_size(name) (range_max(name) - range_min(name) + 1)

@f define_var integer
/* The macro define_var defines variables of the form ind_x_n which are equated with the args.
   to define_var represented by dots. The args. are names of the dimensions of the array x or
   name of the length of character variable x. The macro defines a variable type_x (an integer
   which tells us whether it is real(2), integer(1) or character(0) ). x is substituted by the name of
   the array/character variable and n is the no. of the arg. represented by the dots. rank_x is the
   no. of args represented by the dots. */

// Suppose we have defined :
// define_dimen(a,0,10)
// define_dimen(b,1,10)
// define_dimen(c,5,20)
// define_var(temp,real,a,b,c)

// we get ind_temp_1 = a, ind_temp_2 = b, ind_temp_3 = c
/* ind_x_m equals the name of the last argument and ind_temp_m =c in the above example. */

@m define_var(x, type,...)
$ASSERT (#0 ≤ MAX_DIM ∧ #0 ≥ 0)$ASSERT (type ≥ CHAR ∧ type ≤ FLOAT)$ASSERT (type ≠
CHAR ∨ #0 ≥ 1)$IF (_type==_CHAR,_$ASSERT(range_min(#1)==_1\
),_)$DEFINE(type_##x##type)$DEFINE(rank_##x##0)$IF (#0>=_1,\ \
$DEFINE(ind_##x##_1##1),)$IF (#0>=_2,_$DEFINE(ind_##x##_2##2),)$IF (#0>=_3,\ \

```

```

    )$DEFINE(ind_##x##_3#3),)$IF (#0>=4,)$DEFINE(ind_##x##_4#4),)$IF (#0>=5,\n
    )$DEFINE(ind_##x##_5#5),)$DEFINE(ind_##x##_m$IFCASE(#0,0,#1,#2,#3,#4,#5))

@m define_varp(x, type,...)
$ASSERT((type ≡ CHAR ∧ #0 ≥ 2) ∨ (type ≠ CHAR ∧ #0 ≥ 1)) define_var(x, type, #.)

/* Return type declaration (including character size) */

@if FORTRAN77
@m char_spec(size) *(size)
#else
@m char_spec(size) (len = size)
#endif

@m var_type(x) $IFCASE(type##x,character,integer,real)$IF(type##x==CHAR,\n
    uchar_spec(range_max(ind##x##_1)),)

@m var_type_star(x)
$IFCASE(type##x,character,integer,real)$IF(type##x==CHAR,char_spec(*),)

/* The explicit array shape, e.g., (0 : 1, 3 : 10) */
@m array_shape(x) $IF(type##x==CHAR,$IFCASE(rank##x,,,\n
    range(ind##x##_2),range(ind##x##_2),\n
    range(ind##x##_3),range(ind##x##_2),range(ind##x##_3),\n
    range(ind##x##_4),range(ind##x##_2),range(ind##x##_3),\n
    range(ind##x##_4),range(ind##x##_5)),$IFCASE(rank##x\n
    ,range(ind##x##_1),range(ind##x##_1),\n
    range(ind##x##_2),range(ind##x##_1),range(ind##x##_2),\n
    range(ind##x##_3),range(ind##x##_1),range(ind##x##_2),\n
    range(ind##x##_3),range(ind##x##_4),range(ind##x##_1),\n
    range(ind##x##_2),range(ind##x##_3),range(ind##x##_4),range(ind##x##_5)))

/* The shape with the last bound unspecified, e.g., (0 : 1, 3 : *) */
@if FORTRAN77
@m array_shape_star(x) $IF(type##x==CHAR,$IFCASE(rank##x,,,\n
    range_star(ind##x##_2),range(ind##x##_2),\n
    range_star(ind##x##_3),range(ind##x##_2),range(ind##x##_3),\n
    range_star(ind##x##_4),range(ind##x##_2),range(ind##x##_3),\n
    range(ind##x##_4),range_star(ind##x##_5)),$IFCASE(rank##x\n
    ,range_star(ind##x##_1),range(ind##x##_1),\n
    range_star(ind##x##_2),range(ind##x##_1),range(ind##x##_2),\n
    range_star(ind##x##_3),range(ind##x##_1),range(ind##x##_2),\n
    range(ind##x##_3),range_star(ind##x##_4),range(ind##x##_1),\n
    1),range(ind##x##_2),range(ind##x##_3),range(ind##x##_4),\n
    range_star(ind##x##_5)))

/* The shape with the last size set to 1. This is because IBM's xlf doesn't like a pointer array to\n
have a starred last dimension */

@m array_shape_stara(x) $IF(type##x==CHAR,$IFCASE(rank##x,,,\n
    range_stara(ind##x##_2),range(ind##x##_2),\n
    range_stara(ind##x##_3),range(ind##x##_2),range(ind##x##_3),\n
    range_stara(ind##x##_4),range(ind##x##_2),range(ind##x##_3),\n
    range(ind##x##_4),range_stara(ind##x##_5)),$IFCASE(rank##x\n
    ,range_stara(ind##x##_1),range(ind##x##_1),\n
    range_stara(ind##x##_2),range(ind##x##_1),range(ind##x##_2),\n
    range_stara(ind##x##_3),range(ind##x##_1),range(ind##x##_2),\n
    range(ind##x##_3),range_stara(ind##x##_4),range(ind##x##_1),\n
    1),range(ind##x##_2),range(ind##x##_3),range(ind##x##_4),\n
    range_stara(ind##x##_5)))

```

```

    1), range(ind_##x##_2), range(ind_##x##_3), range(ind_##x##_4), \
    range_stara(ind_##x##_5)))))

@#endif

/* A Fortran-90 style rank, e.g., (:,:) */

@if FORTRAN90
@m array_rank(x)
$IF (type_##x==CHAR, $IFCASE(rank_##x,,(:,:),(::,:),
                               (:,:,:,:)), $IFCASE(rank_##x,
                               (:,:,:), (:,:,:,:), (:,:,:,:),
                               (:,:,:,:,:,:)))

@#endif

/* Comma separated shape, e.g., 0, 1, 3, y. Character length is the first argument */

@if FORTRAN90
@m array_arglistc(x,y)
$IF (type_##x==CHAR, $IFCASE(rank_##x,,range_max(ind_##\
x##_1), range_max(ind_##x##_1) COMMA range_min(ind_##x##_2) COMMA y, \
range_max(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA range_min(ind_##x##_3) COMMA y, \
range_max(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_3) COMMA y, \
range_max(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_4) COMMA y, \
range_max(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_3) COMMA y, \
range_max(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_4) COMMA y, \
range_max(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_3) COMMA y, \
range_min(ind_##x##_1) COMMA y, rangea(ind_##x##_1) COMMA range_min(ind_##x##_2) COMMA y, \
rangea(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA range_min(ind_##x##_3) COMMA y, \
rangea(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_4) COMMA y, \
rangea(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_3) COMMA y, \
rangea(ind_##x##_1) COMMA rangea(ind_##x##_2) COMMA rangea(ind_##x##_4) COMMA y))

@m array_arglist(x) array_arglistc(x, range_max(ind_##x##_m))

@m var_typea(x)
paste($IFCASE(type_##x,c,i,r), $IF (type_##x==CHAR,$EVAL(rank_##x-1),rank_##x))

@#endif

@f declare_var integer
/* declare_var (x) declares the variable x. It may be a character, real or integer and can have 0
   to MAX_DIM dimensions. For this to work, the values of ind_x_n and corresponding range must
   be known through the define_dimen and define_var macros explained earlier. */

// suppose we have
// define_dimen(len_junk, 10)
// define_dimen(len_junk_array, 32)
// define_dimen(dim_junk_array, 1, 5)
// define_var(junk, CHAR, len_junk)
// define_var(junk_array, CHAR, len_junk_array, dim_junk_array)
// declare_var(junk)
// declare_var(junk_array)
// we get:
// character * 10 junk
// character * 32 junk_array(1:5)

// NB the syntax of the $IF statement is :

```

```

/* $IF (expression,true,false). note the position of the commas carefully. They separate the true
   and false parts. */

@m declare_var(x) var_type(x)xarray_shape(x)

@f declare_varp integer

/* declare_varp(x) is similar to declare_var(x), but it declares a pointer to an array x . It is
   necessary to declare a pointer to an array for dynamic allocation. So the last upper dimension is
   not fixed. suppose we have : */

// define_dimen(len_junk_array, 32)
// define_dimen(lower_junk_array, 1)
// define_var(junk_array, CHAR, len_junk_array, lower_junk_array)
// declare_varp(junk_array)
// then we get:
// character * 32 junk_array(1 : *)
// pointer(ptr_junk_array, junk_array)

/* The last statement comes from the macro ptr_decl(x) which is explained later. */

@if FORTRAN77
@if ¬IBM
@m declare_varp(x) ptr_decl(x)
var_type(x)xarray_shape_star(x)
#else
@m declare_varp(x) ptr_decl(x)
var_type(x)xarray_shape_stara(x)
#endif
#else
@m declare_varp(x) var_type(x), dimension array_rank(x), pointer :: x
#endif

/* array_size(x) is the total size of an array x. It is given as the product of the ranges of the
   dimensions. The range is defined by a macro range_size and is basically - dimU - dimL +1 */

@m array_size(x)
$IFCASE (rank##x,1,(range_size(ind##x##_1)),,,
          (range_size(ind##x##_
-1)*range_size(ind##x##_2)),,,,
          (range_size(ind##x##_
-1)*range_size(ind##x##_2)*range_size(ind##x##_3)),,,,
          (range_size(ind##x##_
-1)*range_size(ind##x##_2)*range_size(ind##x##_3)*range_size(ind##x##_4)),\,
          (range_size(ind##x##_1)*range_size(ind##x##_2)*range_size(ind##x##_3)*range_size(ind##x##_4)*range_size(ind##x##_5)))
/* array_unit(x) is the size calculated by leaving out the last dimension of the array x. This
   was previously set for use only with FORTRAN77, but don't know why. Using this with
   FORTRAN90 rather than developing analogous capability (via size function?). */

@m array_unit(x)
$IFCASE (rank##x,,1,,,
          (range_size(ind##x##_1)),,,,
          (range_size(ind##x##_
-1)*range_size(ind##x##_2)),,,,
          (range_size(ind##x##_
-1)*range_size(ind##x##_2)*range_size(ind##x##_3)),,,,
          (range_size(ind##x##_
-1)*range_size(ind##x##_2)*range_size(ind##x##_3)*range_size(ind##x##_4)))

/* array_sizev(x, y) gives the size of an array x if its last dimension (upper limit) is y. This can
   be used to calculate the new size of an array x if it's last dimension is changed. It's basically the
   product of the size calculated by leaving out the last dimension (array_unit(x)) and the new
   size of the last dimension which is just = (dimnU -dimnL +1) where dimnU is y and dimnL is
   known.( lower limit of last dimension given by range_min(ind_x_m) */

```

```

@if FORTRAN77
@m array_sizev(x,y) array_unit(x) * ((y) - range_min(ind_##x##_m) + 1)
#endif

/* var_beg(x) gives the initial element of an array */

@m var_beg(x) x$IF (type_##x==CHAR, $IFCASE(rank##x,,,,
    range_min(ind##x##_2)), range_min(ind##x##_2),
    range_min(ind##x##_3)), range_min(ind##x##_2),
    range_min(ind##x##_3), range_min(ind##x##_4)), range_min(ind##x##_2),
    range_min(ind##x##_3), range_min(ind##x##_4), range_min(ind##x##_5))), \
    $IFCASE(rank##x,, (range_min(ind##x##_1)), \
    (range_min(ind##x##_1), range_min(ind##x##_2)), \
    (range_min(ind##x##_1), range_min(ind##x##_2),
    range_min(ind##x##_3)), (range_min(ind##x##_1),
    range_min(ind##x##_2), range_min(ind##x##_3),
    range_min(ind##x##_4)), \
    (range_min(ind##x##_1), range_min(ind##x##_2),
    range_min(ind##x##_3), range_min(ind##x##_4)))

```

[/Users/dstotler/degas2/src/array.hweb] Array slices for passing to subroutines. I think we only need a full one-dimensional slice of numeric arrays and that's what's given here.

```

"classes.f" 50.3 ≡
#if FORTRAN77
@m slice1(x, dims) x dims range_min(ind##x##_1)
#else
@m slice1(x, dims) x dims :
#endif

```

```
[/Users/dstotler/degas2/src/array.hweb] Macros to simplify memory allocation

/* ptr_decl(x) declares a pointer to an array x and calls it ptr_x where x is the name of the array. */
"classes.f" 50.4 ≡
@if FORTRAN77
@f ptr_decl integer
@m ptr_decl(x) pointer(ptr##x, x)
@m mem_assign(x,p) ptr##x = p
#else
@m mem_assign(x,p) xASSIGNp
#endif

/* real_alloc(x, n) allocates memory for a real array x of size n .real_realloc(x, on, n) reallocates
   memory for an real array x which has a size on but needs a size n. Once the array has been
   used, its memory can be freed using real_free(x, n) */

@if FORTRAN77
@m real_alloc(x,n) ptr##x = mem_alloc(n)
@m real_realloc(x,on,n) ptr##x = mem_realloc(ptr##x, on, n)
@m real_free(x,n) call mem_free(ptr##x, n)
#endif

/* similar macros for allocating, reallocating and freeing memory for integers. they use the same
   function calls, only the size is different, this is handled by using int_mem(n) instead of just
   n which does the right thing */

@if FORTRAN77
@m int_alloc(x,n) real_alloc(x, int_mem(n))
@m int_realloc(x,on,n) real_realloc(x, int_mem(on), int_mem(n))
@m int_free(x,n) real_free(x, int_mem(n))
#endif

/* similar macros for allocating, reallocating and freeing memory for characters. they use the
   same function calls, only the size is different, this is handled by using char_mem(n) instead of
   just n which does the right thing */

@if FORTRAN77
@m char_alloc(x,n) real_alloc(x, char_mem(n))
@m char_realloc(x,on,n) real_realloc(x, char_mem(on), char_mem(n))
@m char_free(x,n) real_free(x, char_mem(n))
#endif

/* var_alloc(x), var_realloc(x, on, n) and var_free(x) are generalisations of the above macros
   which were meant for a specific type. Here, the macro does the right thing depending on whether
   the allocation, reallocation or freeing has to be done for a real, integer or character. It finds out
   the size of the array too, using the macro array_size(x). note the use of array_sizev(x, y) to get
   the size of x with the last dimension set to n and on */

@if FORTRAN77
@m var_alloc(x) $IFCASE (type##x, @char_alloc(x, array_size(x)), @int_alloc(x, \
array_size(x)), @real_alloc(x, array_size(x)))
#else
@m var_alloc(x) xASSIGN paste(mem_alloc_, var_typea(x))(array_arglist(x), #x)
#endif

@if FORTRAN77
```

```
@m var_realloc(x, on, n) $IFCASE (type_##x, char_realloc(x, array_sizev(x, on), \
array_sizev(x, n)), int_realloc(x, array_sizev(x, on), array_sizev(x, n)), \
real_realloc(x, array_sizev(x, on), array_sizev(x, n)))  
@#else  
    @m var_realloc(x, on, n) xASSIGN paste(mem_realloc_, var_typea(x))(x, array_arglistc(x, on), n, #x)  
@#endif  
  
@#if FORTRAN77  
    @m var_free(x) $IFCASE (type_##x, char_free(x, array_size(x)), int_free(x, \
array_size(x)), real_free(x, array_size(x)))  
@#else  
    @m var_free(x) call paste(mem_free_, var_typea(x))(x, array_arglist(x), #x)  
@#endif
```

[/Users/dstotler/degas2/src/array.hweb] Macros to accomodate growing arrays.

```
"classes.f" 50.5 ≡
@m mem_inc 100

/* mem_size(x) returns the size in the nearest unit of mem_inc for a given size x. eg.. mem_size(1), mem_size(2) ... mem_size(100) all return 100, but mem_size(101) ...mem_size(200) = 200 etc.
*/
@m mem_size(x) (int(((x) + mem_inc - 1) / mem_inc) * mem_inc) // [x/mem_inc]mem_inc

/* array_sizea(x) gives the size of an array x using the nearest multiple of mem_inc as the last range. for eg.. if mem_inc =5, then mem_size(1) to mem_size(5) = 5, mem_size(6) to mem_size(10) = 10 etc. now if x is x(1:5), array_sizea(x) is 1 × 5 = 5 and if x is x(1:6), it is 1 × 10 = 10. array_sizeb(x) gives the previous size of x using the nearest multiple of mem_inc as the last range. it is just the mem_size(x) using the last index minus mem_inc. e.g.. if x is x(1:6), array_sizeb(x) = 1 × (mem_size(6) - 5) = 1 × (10 - 5) =5 array_sizec(x, y) gives the size of an array x using the nearest multiple of mem_inc as the last range (same as array_sizea(x) so far). It uses the upper limit y for the last index instead of the original value.*/
@m last_sizea(x) mem_size(range_size(ind_##x##_m))
@m last_sizeb(x) (mem_size(range_size(ind_##x##_m)) - mem_inc)
@m last_sizec(x, y) mem_size((y) - range_min(ind_##x##_m) + 1)

@m last_ubound(x) range_max(ind_##x##_m)
@m last_bounda(x) (last_sizea(x) + range_min(ind_##x##_m) - 1)
@m last_boundb(x) (last_sizeb(x) + range_min(ind_##x##_m) - 1)
@m last_boundc(x, y) (last_sizec(x, y) + range_min(ind_##x##_m) - 1)

@if FORTRAN77
@m array_sizea(x) array_unit(x) * last_sizea(x)
@m array_sizeb(x) array_unit(x) * last_sizeb(x)
@m array_sizec(x, y) array_unit(x) * last_sizec(x, y)
#endif

/* var_realloca(x) reallocates the size of an array x if it's growing in increments of 1, but increments it in steps of mem_inc when it's size has just exceeded the limit. since the value of the range at this point is like mem_inc *n +1 (n is an integer), mem_size ( range_size(indx_m) gives the next higher multiple of its size. This is the new size. The old size is this value minus mem_inc which is what the macros array_sizea(x) and array_sizeb(x) return.*/
@if FORTRAN77
@m var_realloca(x) if (mod(range_size(ind_##x##_m), mem_inc) ≡ 1) then
$IFCASE (type_##x, char_realloc(x, array_sizeb(x), array_sizea(x)), \
         int_realloc(x, array_sizeb(x), array_sizea(x)), real_realloc(x, \
         array_sizeb(x), array_sizea(x)))
end if
@else
@m var_realloca(x) if (mod(range_size(ind_##x##_m), mem_inc) ≡ 1) then
var_realloc(x, last_boundb(x), last_bounda(x))
end if
#endif

/* var_reallocb(x) sets the size of the array to the actual size at that time given by array_size , if the range size of the last index is not a multiple of mem_inc (which means the growth has saturated ) */
@if FORTRAN77
```

```

@m var_reallocb(x) if (mod(range_size(ind_##x##_m), mem_inc) ≠ 0) then
    $IFCASE (type_##x, char_realloc(x, array_sizea(x), array_size(x)), \
              int_realloc(x, array_sizea(x), array_size(x)), real_realloc(x, \
              array_sizea(x), array_size(x)))
end if
@#else
@m var_reallocb(x) if (mod(range_size(ind_##x##_m), mem_inc) ≠ 0) then
    var_realloc(x, last_ubounda(x), last_ubound(x))
end if
@#endif

/* var_reallocc(x, on, n) reallocates the size of an array x if the sizes of x corresponding to the
   old value of upper limit of the last dim. (on) rounded off to the nearest multiple of mem_inc
   and the size of x corresponding to the new value of upper limit of the last dim. (n) rounded off
   to the nearest multiple of mem_inc are not the same. */

@if FORTRAN77
@m var_reallocc(x, on, n)
if (mem_size((on) - range_min(ind_##x##_m) + 1) ≠ mem_size((n) - range_min(ind_##x##_m) + 1))
    then
$IFCASE (type_##x, char_realloc(x, array_sizec(x, on), array_sizec(x, n)), \
          int_realloc(x, array_sizec(x, on), array_sizec(x, n)), real_realloc(x, \
          array_sizec(x, on), array_sizec(x, n)))
end if
@#else
@m var_reallocc(x, on, n)
if (mem_size((on) - range_min(ind_##x##_m) + 1) ≠ mem_size((n) - range_min(ind_##x##_m) + 1))
    then
    var_realloc(x, last_uboundc(x, on), last_uboundc(x, n))
end if
@#endif

```

[/Users/dstotler/degas2/src/array.hweb] Define package functions.

```
/* Categories of package definition macros */

"classes.f" 50.6 ≡
@m PK_DUMMY 0
@m PK_DIMEN 1
@m PK_VAR 2
@m PK_VARP 3
@m PK_VARLOCAL 4
@m PK_CAT_MIN PK_DUMMY
@M PK_CAT_MAX PK_VARLOCAL

@m register(pk,n,x,cat) _register(pk, n, x, cat)
@m _register(pk,n,x,cat) $DEFINE (VAR_##pk##_##n_ux)$DEFINE (CAT_##pk##_##n_cat)

@m package_init(pk) $DEFINE (COUNT_##pk_0)$DEFINE (pk##_common_process_pk(pk,\ 
    common))$DEFINE (pk##_module_process_pk(pk,\ 
    module))$DEFINE (pk##_ncdef(fileid)_process_pk(pk,ncdef,\ 
    fileid))$DEFINE (pk##_ncwrite(fileid)_process_pk(pk,\ 
    ncwrite,fileid))$DEFINE (pk##_ncread(fileid)_proces\
    s_pk(pk,ncread,fileid))$DEFINE (pk##_ncdecl_process_pk(pk,\ 
    ncdecl))$DEFINE (pk##_pvmput_process_pk(pk,\ 
    pvmput))$DEFINE (pk##_pvmget_process_pk(pk,\ 
    pvmget))$DEFINE (pk##_mpibcast_process_pk(pk,\ 
    mpibcast))$DEFINE (pk##_mpibcastna_process_pk(pk,\ 
    mpibcastna))$DEFINE (pk##_mpisend(dest,tag)_process_pk(pk,mpisend,dest,\ 
    tag))$DEFINE (pk##_mpireceive(source,tag)_process_pk(pk,mpireceive,source,tag))

@m package_end(pk)

@m define_dimen_pk(pk,name,...)
    define_dimen(name, #.) $INCR (COUNT_##pk)register(pk, COUNT_##pk, name, PK_DIMEN)

@m define_dummy_pk(pk,name,type,...)
    define_var (name, type, #.) $INCR (COUNT_##pk)register(pk, COUNT_##pk, name, PK_DUMMY)

@m define_var_pk(pk,name,type,...)
    define_var (name, type, #.) $INCR (COUNT_##pk)register(pk, COUNT_##pk, name, PK_VAR)

@m define_varp_pk(pk,name,...)
    define_varp(name, #.)$INCR (COUNT_##pk)register(pk, COUNT_##pk, name, PK_VARP)

@m define_varlocal_pk(pk,name,type,...) define_var (name, type,
#.) $INCR (COUNT_##pk)register(pk, COUNT_##pk, name, PK_VARLOCAL)
```

[/Users/dstotler/degas2/src/array.hweb] Iterating macros. These iterate over all the entities in a package, invoking a macro for entity.

```
"classes.f" 50.7 ≡
@m process_pk(pk, module, ...)
$DEFINE (TEMPCOUNT_0) start_##module(pk, #.) iterate_pk(pk, module, #.) end_##module(pk, #.)
// The "*" here is significant

@m* iterate_pk(pk, module, ...) $INCR(TEMPCOUNT)$IF(TEMPCOUNT<=COUNT_##pk,\n    do_##module(pk,paste(VAR##pk##_,TEMPCOUNT),paste(CAT##pk##_,\n    TEMPCOUNT),#.) iterate_pk(pk,module,#.) )
@#if FORTRAN77
@m start_common(pk)
$DEFINE (COMMON_C_0)$DEFINE (COMMON_I_0)$DEFINE (COMMON_R_0)$DEFINE (COMMON_P_0)

@m end_common(pk)
$IF(COMMON_C>0, save/pk##_com_c/;, )$IF(COMMON_I>0, save/pk##_com_i/;, )
$IF(COMMON_R>0, save/pk##_com_r/;, )$IF(COMMON_P>0, save/pk##_com_p/;, )

@m do_common(pk, var, cat) $ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,\n    declare_var(var); $IFCASE(paste(type_,var),\n        common/pk##_com_c/_var; $INCR(COMMON_C), common/pk##_\n        com_i/_var; $INCR(COMMON_I), common/pk##_com_r/_var; $INCR(COMMON_R)\n    ), declare_varp(var); common/pk##_com_p/_paste(ptr_,\n        var); $INCR(COMMON_P), declare_var(var); $IFCASE(paste(type_\n        ,var), common/pk##_com_c/_var; $INCR(COMMON_C),\n        common/pk##_com_i/_var; $INCR(COMMON_I), common/pk##_\n        com_r/_var; $INCR(COMMON_R)), common/pk##_\n        com_r/_var; $INCR(COMMON_R))
@#else
@m start_common(pk)
@m end_common(pk) use pk##_mod
@m do_common(pk, var, cat)

@#endif

@#if FORTRAN90
@if 0 // This version only outputs module, and end module if there's a body.
// This break things with MPI on non-MPI builds.
// This follow version fixes this.
@m start_module(pk) $DEFINE (MODULE_SIZE_0)

@m end_module(pk) $IF(MODULE_SIZE>0, end_module_pk##_mod;, )
@m do_module(pk, var, cat) $ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,\n    $IF(MODULE_SIZE==0, module_pk##_mod; implicit\n        t_none_f90mod; save;, ) declare_var(var); $INCR(MODULE_SIZE),\n        $IF(MODULE_SIZE==0, module_pk##_mod; implicit_none_f90mod; save;, \n            declare_varp(var); $INCR(MODULE_SIZE),\n            $IF(MODULE_SIZE==0, module_pk; implicit_none_f90mod; save;, \n                declare_var(var); $INCR(MODULE_SIZE),\n                )
@#else
@m start_module(pk) module_pk##_mod
implicit_none_f90mod save

@m end_module(pk) end module_pk##_mod
```

```

@m do_module(pk, var, cat) $ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,\n
                                     declare_var(var);,declare_varp(var);,\n
                                     declare_var(var);,)\n
#endif\n
#ifndef\n
@m start_module(pk)\n
@m end_module(pk)\n
@m do_module(pk, var, cat)\n
#endif\n
@m start_ncdef(pk, fileid)\n
@m do_ncdef(pk, var, cat, fileid)\n
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,,\n
                                                     dimen_def(fileid,var);,var_def(fileid,var);,\n
                                                     var_def(fileid,var);,)\n
@m end_ncdef(pk, fileid)\n
@m start_ncdecl(pk)\n
@m do_ncdecl(pk, var, cat) $ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,\n
                                     integer_dimid(var);,integer_dimid(var);,\n
                                     integer_dimid(var);,)\n
@m end_ncdecl(pk)\n
@m start_ncwrite(pk, fileid)\n
@m do_ncwrite(pk, var, cat, fileid)\n
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,,,,\n
                                                     var_write(fileid,var);,var_write(fileid,var);,,)\n
@m end_ncwrite(pk, fileid)\n
@m start_ncread(pk, fileid)\n
@m do_ncread(pk, var, cat, fileid)\n
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,,,\n
                                                     dimen_inq(fileid,var);,var_inq(fileid,var);,var_read(fileid,\n
                                                     var);,var_inq(fileid,var);,var_alloc(var);,var_read(fileid,var);,,)\n
@m end_ncread(pk, fileid)\n
@m start_pvmput(pk)\n
@m do_pvmput(pk, var, cat) $ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,\n
                                     ,,,pvm_put(var);,pvm_put(var);,,)\n
@m end_pvmput(pk)\n
@m start_pvmget(pk)\n
@m do_pvmget(pk, var, cat)\n
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat,,,,\n
                                                     pvm_get(var);,var_alloc(var);,pvm_get(var);,,)\n
@m end_pvmget(pk)\n
@m start_mpibcast(pk)

```

```

@if MPI
@m do_mpibcast(pk, var, cat)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, mpi_broadcast(var);, if (mpi_rank != mpi_degas2_root) then; var_ \
alloc(var); endif; mpi_broadcast(var);, )
#else
@m do_mpibcast(pk, var, cat) $ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, \
, , , , , , )
#endif

@m end_mpibcast(pk) //
// This is the same as the above, but skips the
// allocation step on the slaves; to be used
// for iterative calculations.
//
@m start_mpibcastna(pk)

@if MPI
@m do_mpibcastna(pk, var, cat)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, mpi_broadcast(var);, mpi_broadcast(var);, )
#else
@m do_mpibcastna(pk, var, cat)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, , , , , )
#endif

@m end_mpibcastna(pk)
@m start_mpisend(pk, dest, tag)

@if MPI
@m do_mpisend(pk, var, cat, dest, tag)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, mpi_send(var, dest, tag);, mpi_send(var, dest, tag);, )
#else
@m do_mpisend(pk, var, cat, dest, tag)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, , , , , )
#endif

@m end_mpisend(pk, dest, tag)
@m start_mpireceive(pk, source, tag)

@if MPI
@m do_mpireceive(pk, var, cat, source, tag)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, mpi_receive(var, source, tag);, var_alloc(var); mpi_receive(var, source, tag);, )
#else
@m do_mpireceive(pk, var, cat, source, tag)
$ASSERT(cat ≥ PK_CAT_MIN ∧ cat ≤ PK_CAT_MAX)$IFCASE(cat, , , , , )
#endif

@m end_mpireceive(pk, source, tag)

```

```
@m do_netcdfa(pk, var, cat)
$ASSERT(cat >= PK_CAT_MIN & cat <= PK_CAT_MAX)$IFCASE(cat,
    netcdf0(var);,
    netcdf1(var);, netcdf2(var);, netcdf3(var);,
```

51 Description of array definition macros

[/Users/dstotler/degas2/src/array.hweb]

These macros aid in defining array operations. The way the macros are used is outlined below; they are largely invoked in other header files. The primary exceptions to this rule are the pointer-related macros, *var_alloc*, *var_realloc*, and *var_free*. Perhaps the *best* way to see what these macros do is to examine the FORTRAN code produced by the WEB preprocessor.

The macros used most frequently are those associated with “package definition”:

package_init(CL) This would be the first macro executed for the class with abbreviation *CL*. It begins the package definition process by telling WEB to define “variables” (really they are macros the names of which are defined “on-the-fly” using the class abbreviations) like *CL_common_b*, *CL_ncreada*, and *CL_pvmput*. Then, in the actual WEB code, the programmer uses these phrases to symbolize a complete set of declarations or definitions for this class. This set of definitions, etc. are built up by subsequent macro calls made in the header file as the various dimensions and variables are defined. It is worth going through these macros or “variables” one by one:

CL_common_a Produces common blocks containing scalar (FORTRAN) variables. This includes a “save” statement for each one. The suffix “_a” is used to represent scalars; likewise, “_b” is used for “dimensions” (see below) and “_c” for arrays. Note that typically one finds in the header file for the class a macro *CL_common* which concatenates one or more of these macros into a single one.

CL_ncdef Handles the dimensions and variable definitions required for the “definition” phase of creating a new netCDF file.

CL_ncwrite Writes all of the netCDF variables in class *CL* into the netCDF file specified.

CL_ncread Reads all of the netCDF variables in class *CL* from the specified netCDF file into the FORTRAN code.

CL_ncdecl Declares for the FORTRAN compiler the integer identifiers netCDF uses to identify the various dimensions and variables in the class *CL*.

CL_pvmput Associated with PVM...

CL_pvmget Associated with PVM...

define_dimen_pk(CL, dimname,...) This defines a dimension, i.e., assigns a name, *dimname*, to an integer range. This macro is primarily required only for the netCDF-related macros listed above; the resulting “dimensions” do not appear in the final FORTRAN code except as numbers in these netCDF calls.

define_var_pk(CL, varname, type,...) Defines a FORTRAN variable *varname*. This can be a scalar, array, or a character variable. *varname* is also added to the corresponding netCDF and PVM macro definitions.

define_varp_pk(CL, varname,...) Defines a FORTRAN pointer to be associated with *varname*. Again, the macro appends entries to all of the corresponding netCDF and PVM macro definitions. Note that in the case of *CL_ncread*, the macro *var_alloc(varname)* (see below) is included. In general, a pointer should be used for variables with dimensions which are not known at compile time.

These “package” macros are largely built up on the other macros defined in this header file. The most significant of these are:

define_dimen (*dimname*, *min*, *max*) Defines a dimension range *dimname* as running from *min* to *max*; produces no code.

define_var (*array*, *type*, *dimname1*, *dimname2*,...) Defines *array* of type *type* (*CHAR*, *INT*, or *FLOAT*) as having first dimension *dimname1* (as declared with a **define_dimen** or **define_dimen_pk** macro), and so on. The number of dimensions in the argument list is the rank of *array*. This macro is used for scalars as well (rank 0). Again, no code is produced by this macro.

declare_var (*varname*) Once a variable *varname* is defined, this macro is used to actually declare it in the FORTRAN code; the result is a type (including dimensions for arrays and string length for character strings) declaration statement. The variable type is handled automatically by the macro. Note that for character strings, the first dimension provides the length of each element of the array. This makes interaction with netCDF easier where only single character variables are permitted.

declare_varp (*varname*) Same as **declare_var**, but also uses **ptr_decl** to actually declare the pointer variable itself (see below).

array_size (*arrayname*) Gives the size of *arrayname*, i.e., the total number of elements in an array.

array_unit (*arrayname*) Number of elements in one “slice” (i.e., one unit of the last array index) of the array *arrayname*.

array_sizev (*arrayname*, *newmax*) Gives the size of the array *arrayname* if the upper limit of the last index were set to *newmax*.

ptr_decl (*arrayname*) Declares a pointer names *ptr_arrayname* to be associated with the array *arrayname*.

var_alloc (*arrayname*) Allocates space for array *arrayname*; this assumes that all of its dimensions have been defined.

var_realloc (*arrayname*, *oldmax*, *newmax*) Is used when there is an increase in the last dimension of an array *arrayname*; *oldmax* is the old value for the upper limit and *newmax* is the new value.

var_free (*arrayname*) Deallocates the memory associated with the array *arrayname*.

var_realloca (*arrayname*) Used to reallocate space when the last dimension of array *arrayname* grows by one. The allocation will only take place every *mem_inc* (defined in this file) steps.

var_reallocb (*arrayname*) Once *arrayname* has grown to its final size (via repeated invocations of **var_realloca**), this macro is used to repack the array in the smallest possible space.

var_reallocc (*arrayname*, *oldmax*, *newmax*) Used to reallocate space when the last dimension of array *arrayname* is growing by steps other than one; *oldmax* and *newmax* are the old and new values of the upper limit of the last dimension, respectively.

If one wishes to understand more completely these macros, it is useful to be familiar with some of the predefined WEB macros. Throughout the macro definitions also be aware of which quantities are the arguments of the macros, which are other macros, and which are strings meant to appear in the final FORTRAN code. The binary operators used in the macros are the same as those used in C; see the section entitled “Expression Evaluation” in the WEB manual if these are unfamiliar to you.

@f Denotes a formatting declaration for WEB; affects only the typeset output.

@m A macro definition.

Is the concatenation or paste operator. It is typically used to paste a string argument to a macro into a string inside the macro definition. For example, if we used somewhere **ptr_decl**(*foo*), the definition *ptr##x* would be expanded as *ptr_foo* in the FORTRAN code.

... When an ellipsis appears in an argument list, it means that this macro has a variable number of arguments. The ellipsis may be preceded by specified arguments as well.

#0 When the macro has a variable number of arguments, ‘#0’ expands during WEB processing into the actual number of variable arguments used in this particular invocation of the macro; note that this number does not include any arguments explicitly specified at the beginning of the macro argument list.

#1 Also for a variable number of arguments, ‘#n’ with $n > 0$ is replaced during WEB processing by the n -th argument in the actual invocation of the macro. Again, this does not count any explicit arguments in the macro definition.

\$DEFINE This WEB macro is typically used to assign a particular value, usually obtained from a macro argument, to a local (i.e., to the preprocessor) variable, the name of which is also likely constructed using a macro argument. The other macros whose names begin with an underscore such as “\$IFCASE” are described in the WEB manual; their usage should be obvious. The most effective use of the “\$DEFINE” macro arises in the construction lengthy package declarations such as *pk_ncdecl* where *pk* represents one of the two letter class designations. Through repeated use of “\$DEFINE” and “\$UNDEF”, these macros are built up piece by piece during the execution of other macros such as *define_dimen_pk* and *define_var_pk*.

52 Macros used to read and write netCDF files

[/Users/dstotler/degas2/src/netcdf.hweb]

[/Users/dstotler/degas2/src/netcdf.hweb] Macro definitions from `netcdf.inc`

\$Id: fe54a835bbe9b58a847036ee5eed0df5c9fe5d29 \$

"`classes.f`" 52.1 ≡

@m `nccreate nccre` // Longer versions of subroutine names

@m `ncopen ncopn`

@m `ncredef ncredf`

@m `ncendef ncendf`

@m `ncclose nclos`

@m `ncattput ncapt`

@m `ncattputc ncaptc`

@m `ncattget ncagt`

@m `ncattgetc ncagtc`

@m `ncdimdef ncddef`

@m `ncvardef ncvdef`

@m `ncvarput ncvpt`

@m `ncvarputc ncvptc`

@m `ncvarget ncvgt`

@m `ncvargetc ncvgtc`

@m `ncdimid ncdid`

@m `ncdiminq ncdinq`

@m `ncvarid ncvid`

@m `ncvarinq ncvinq`

@m `ncvarget ncvgt`

@m `ncvargetc ncvgtc`

@m `NC_CHAR 2`

@m `NC_BYTE 1`

@m `NC_SHORT 3`

@m `NC_LONG 4`

@m `NC_FLOAT 5`

@m `NC_DOUBLE 6`

@#if defined (NETCDFV2)

@m `NC_CLOBBER 11`

@m `NC_NOCLOBBER 15`

@#else

@m `NC_CLOBBER 0`

@m `NC_NOCLOBBER 4`

@m `NF_64BIT_OFFSET 512` // Not available under V. 2.

@m `NF_NETCDF4 4096` // AKA HDF5; V. 4 or later

@#endif

@m `NC_WRITE 1`

@m `NC_NOWRITE 0`

@m `NC_GLOBAL 0`

@m `NC_FATAL 1`

@m `NC_VERBOSE 2`

@m `NC_REAL NC_DOUBLE`

@m `NC_INT NC_LONG`

@m `MAX_NC_NAME 128`

```
[/Users/dstotler/degas2/src/netcdf.hweb] Routines to declare the dimensions.

"classes.f" 52.2 ≡
@f nc_decls integer
@m nc_decls integer nc_dims(MAX_DIM), nc_corner(MAX_DIM), nc_edge(MAX_DIM), nc_stat,
    nc_size, nc_rank, nc_attr, nc_type
character*MAX_NC_NAME nc_dummy
external ncopen, ncvarid, ncdimid, nccreate, ncvardef, ncdimdef, nf_get_vara_text, nf_inq_format
integer ncopen, ncvarid, ncdimid, nccreate, ncvardef, ncdimdef, nf_get_vara_text, nf_inq_format

@m dimid1(name) name##_id
@m dimid(name) dimid1(name)

@m dimen_def(fileid, name) dimid(name) = ncdimdef(fileid, #name, range_size(name), nc_stat)

@m var_def(fileid, name) $IF (rank_##name_>=1, nc_dims(1)=dimid(ind_##nam\
    e##_1);,)$IF (rank_##name_>=2, nc_dims(2)=dimid(ind_##nam\
    e##_2);,)$IF (rank_##name_>=3, nc_dims(3)=dimid(ind_##name##_3);, \
    )$IF (rank_##name_>=4, nc_dims(4)=dimid(ind_##name##_4);,)$IF (rank_##name_>=5, \
    nc_dims(5)=dimid(ind_##name##_5);,) dimid(name) = ncvardef(fileid, #name, \
    $IFCASE (type##name, NC_CHAR, NC_INT, NC_REAL), rank##name, nc_dims, nc_stat)

@m var_write(fileid, name)
$IF (rank_##name_>=1, nc_corner(1)=1; nc_edge(1)=range_size(ind##name##_1);, \
    )$IF (rank_##name_>=2, nc_corner(2)=1; nc_edge(2)=range_size(ind##name##_2);, \
    )$IF (rank_##name_>=3, nc_corner(3)=1; nc_edge(3)=range_size(ind##name##_3);, \
    )$IF (rank_##name_>=4, nc_corner(4)=1; nc_edge(4)=range_size(ind##name##_4);, \
    )$IF (rank_##name_>=5, nc_corner(5)=1; nc_edge(5)=range_size(ind##name##_5);, \
    );,$IF (type##name==CHAR, call_ncvarputc(fileid, dimid(name), \
    nc_corner, nc_edge, name, array_size(name), nc_stat), call_ncvarput(fileid, \
    dimid(name), nc_corner, nc_edge, name, nc_stat))

@m dimen_inq(fileid, name) dimid(name) = ncdimdef(fileid, #name, nc_stat)
call ncdiminq(fileid, dimid(name), nc_dummy, nc_size, nc_stat)
assert(nc_size ≡ range_size(name))

@m var_inq(fileid, name) dimid(name) = ncvarid(fileid, #name, nc_stat)
call ncvarinq(fileid, dimid(name), nc_dummy, nc_type, nc_rank, nc_dims, nc_attr, nc_stat)
$IF (rank##name_>=1, assert(nc_dims(1)==dimid(ind##name##_1));, \
    )$IF (rank##name_>=2, assert(nc_dims(2)==dimid(ind##name##_2));, \
    )$IF (rank##name_>=3, assert(nc_dims(3)==dimid(ind##name##_3));, \
    )$IF (rank##name_>=4, assert(nc_dims(4)==dimid(ind##name##_4));, \
    )$IF (rank##name_>=5, assert(nc_dims(5)==dimid(ind##name##_5));, )

@#if 0 /* These used to be part of var_inq. Removed to avoid tangle problems. */
assert(nc_rank ≡ rank##name)
assert(nc_rank ≥ 0 ∧ nc_rank ≤ MAX_DIM)
assert($IFCASE (type##name, NC_CHAR, NC_INT, NC_REAL) ≡ nc_type)
@#endif

@#if 0
@m var_read(fileid, name)
$IF (rank##name_>=1, nc_corner(1)=1; nc_edge(1)=range_size(ind##name##_1);, \
    )$IF (rank##name_>=2, nc_corner(2)=1; nc_edge(2)=range_size(ind##name##_2);, \
    )$IF (rank##name_>=3, nc_corner(3)=1; nc_edge(3)=range_size(ind##name##_3);, \
    )$IF (rank##name_>=4, nc_corner(4)=1; nc_edge(4)=range_size(ind##name##_4);, \
    )$IF (rank##name_>=5, nc_corner(5)=1; nc_edge(5)=range_size(ind##name##_5);,
```

```
) ; ,)$IF (type_##name==_CHAR,call_ncvargetc(fileid,dimid(name),\
nc_corner,nc_edge,name,array_size(name),nc_stat),call_ncvarget(fileid,\
dimid(name),nc_corner,nc_edge,name,nc_stat))

@#else
@m var_read(fileid,name)
$IF (rank_##name>=1,nc_corner(1)=range_size(ind_##name##_1);, \
)$IF (rank_##name>=2,nc_corner(2)=range_size(ind_##name##_2);, \
)$IF (rank_##name>=3,nc_corner(3)=range_size(ind_##name##_3);, \
)$IF (rank_##name>=4,nc_corner(4)=range_size(ind_##name##_4);, \
)$IF (rank_##name>=5,nc_corner(5)=range_size(ind_##name##_5);, \
)$IF (type_##name==_CHAR,call_ncvarget(fileid,dimid(name),nc_corner,\
nc_edge,name,nc_stat))

@#endif
```

53 Description of netCDF Macros

[/Users/dstotler/degas2/src/netcdf.hweb]

The purpose of these macros is to reduce the amount of effort the user has to expend to read and write netCDF files. In particular, since the WEB preprocessor already knows the type and dimensioning information for every variable in the header files, it can generate the required netCDF function calls on the fly during processing. In order to fully comprehend these macros, it is useful to be familiar with the concepts of netCDF files and some of the details about the routines used to deal with them. Perhaps the most helpful information can be found within the “Info” feature of *Emacs* on the PPPL Theory workstations.

Note that the programmer is not likely to call directly any of the macros defined in this file. Rather, one would be more likely to use the netCDF-related package macros defined in array.hweb. At the same time, a few of the netCDF routines are simple enough that they are called directly. Let us consider a hypothetical class, cl. Here is a typical sequence of subroutine and macro calls used to create and write a new netCDF file:

```
integer fileid          \\\ With other declarations
cl_ncdecl              \\\ Declare netCDF id integers
.
.
.
fileid = nccreate('cl.nc',NC_CLOBBER,nc_stat)  \\\ Enter define mode
call ncattputc(fileid,NC_GLOBAL,'data_version', \\\ Put attributes
$   NC_CHAR,string_length(cl_version),
$   cl_version,nc_stat)
cl_ncdef(fileid)
call ncendef(fileid,nc_stat)
cl_ncwrite(fileid)           \\\ Now in data mode
call ncclose(fileid,nc_stat)
```

The steps used to read an existing file generally look like:

```
integer fileid          \\\ With other declarations
cl_ncdecl              \\\ Declare netCDF id integers
.
.
.
fileid = ncopen('cl.nc',NC_NOWRITE,nc_stat)
call ncattgetc(fileid,NC_GLOBAL,'data_version',
$   cl_version,len(cl_version),nc_stat)      \\\ Get attributes
cl_ncread
call ncclose(fileid,nc_stat)
```

These macros are extremely effective. It is instructive to look at the FORTRAN code generated by the WEB preprocessor to appreciate just how much effort these macros save. For information on these macros, see the documentation associated with the header file array.hweb.

Looking more closely now at the macros defined in this header file, we see that they are minimal extensions of the actual netCDF routines.

dimen_def(fileid, dimname) Obtains the size of dimension *dimname* and constructs the line of FORTRAN required to call *ncdimdef*; that routine actually defines the dimension for file *fileid*. Note that #name in the macro definition instructs WEB to “stringize” the macro argument *name* (for FORTRAN, place in single quotes) .

var_def(fileid, varname) Checks the rank of the variable *varname*, writes its dimensions to the FORTRAN code via the local array *nc_dims*, and then assembles the call to *ncvardef*, including the type declaration, which defines *varname* in file *fileid*.

var_write(fileid, varname) First writes out to the FORTRAN code the length of every “edge” (one for each dimension) of the variable *varname*; this is done using the local arrays *nc_corner* and *nc_edge*. Then, either the netCDF routine *ncvarputc* (for character strings) or *ncvarput* is called to actually write the data to the file.

dimen_inq(fileid, dimname) Creates two lines of code. The first obtains from the netCDF file the value of the identifier for dimension *dimname*; this will be stored in a local integer variable generated by the *dimid(dimname)* macro. The second line of code obtains the size of the dimension using *ndiminq*. This is checked against the expected size.

var_inq(fileid, varname) Generates two lines of code. The first obtains from the netCDF file the value of the identifier for variable *varname*; this will be stored in a local integer variable generated by the *dimid(varname)* macro. The second line of code obtains the rank, type, and size of *varname*. Each of these is checked in turn against the expected values.

var_read(fileid, varname) First writes out to the FORTRAN code the length of every “edge” (one for each dimension) of the variable *varname*; this is done using the local arrays *nc_corner* and *nc_edge*. Then, either the netCDF routine *ncvargetc* (for character strings) or *ncvarget* is called to actually retrieve the data from the file. Note that *ncvargetc* has been replaced by the function *nf_get_vara_text* due to an incompatibility with the library.

54 Arrays needed to handle particle distribution function data

[/Users/dstotler/degas2/src/snapshot_pdf.hweb]

\$Id: 463f391c3b4fee422cb56ee1b14279234ea27b9e \$

[/Users/dstotler/degas2/src/snapshot_pdf.hweb] Contains an external representation of the particle distribution function (pdf). More precisely, this class provides a record of the discrete record of the pdf at the end of a time step (“snapshot”). The principal application is to then sample from the pdf during a subsequent time step in a time dependent run.

[/Users/dstotler/degas2/src/snapshot_pdf.hweb] Indices for entries in the main arrays. This list is intended to mirror exactly the attributes of a “particle”, specifically, those enumerated in **pt_decl**. No active means of enforcing this correspondence has been implemented. The validity of particle data read back into the code for the purpose of restarting a particle trajectory will be checked, however.

```
"classes.f" 54.2 ≡
@m sn_int_pt_sp 1
@m sn_int_pt_test 2
@m sn_int_lc_cell 3
@m sn_int_lc_zone 4
@m sn_int_lc_face 5
@m sn_int_lc_cell_next 6
@m sn_int_lc_zone_next 7
@m sn_int_lc_sector 8
@m sn_int_lc_sector_next 9
@m sn_int_pt_type 10
@m sn_int_pt_author 11

@m sn_pt_int_max 11

@m sn_float_pt_t 1
@m sn_float_pt_w 2
@m sn_float_pt_v1 3
@m sn_float_pt_v2 4
@m sn_float_pt_v3 5
@m sn_float_lc_x1 6
@m sn_float_lc_x2 7
@m sn_float_lc_x3 8

@m sn_pt_float_max 8
```

[/Users/dstotler/degas2/src/snapshot_pdf.hweb] Variable definitions

```
"classes.f" 54.3 ≡
package_init(sn)

define_var_pk(sn, sn_particles_dim, INT)
define_dimen_pk(sn, snapshot_pdf_ind, sn_particles_dim)
define_dimen_pk(sn, sn_pt_float_ind, sn_pt_float_max)
define_dimen_pk(sn, sn_pt_int_ind, sn_pt_int_max)
define_dimen_pk(sn, sn_seed_decimal_ind, ran_c)

define_var_pk(sn, sn_number_particles, INT)
define_var_pk(sn, sn_seed_decimal, CHAR, sn_seed_decimal_ind)

define_varp_pk(sn, sn_particles_float, FLOAT, sn_pt_float_ind, snapshot_pdf_ind)
define_varp_pk(sn, sn_particles_int, INT, sn_pt_int_ind, snapshot_pdf_ind)

package_end(sn)
```

55 Description of particle snapshot data structures

[/Users/dstotler/degas2/src/snapshot_pdf.hweb]

[/Users/dstotler/degas2/src/snapshot.pdf.hweb] Define structures for storing, passing, and reusing particle data. Prefix is *sn*.

sn_number_particles Number of particles contained in the snapshot.

sn_seed_decimal The random number seed at the end of the run in which the snapshot was taken. This is used as the seed in the subsequent time interval to ensure that the random number sequence is different from that of the previous interval.

sn_particles_float_{ip,i_float} Value of float property *i_float* (see index list above) for snapshot particle *ip*.

sn_particles_int_{ip,i_int} Value of integer property *i_int* (see index list above) for snapshot particle *ip*.

56 File names for DEGAS 2

[/Users/dstotler/degas2/src/readfilenames.hweb]

\$Id: 03cbe8a7de6abcb1d605e865cad6b96a308f855 \$

Symbolic names for DEGAS 2 input and output files.

[/Users/dstotler/degas2/src/readfilenames.hweb] indices for the array of filenames

```
"classes.f" 56.1 ≡
@m elementsfile 1
@m backgroundfile 2
@m testfile 3 // No longer used
@m geometryfile 4
@m problemfile 5
@m reactionfile 6
@m speciesfile 7
@m aladinfile 8 // Not being used
@m aladoutfile 9 // Not being used
@m elements_infile 10
@m problem_infile 11
@m reaction_infile 12
@m species_infile 13
@m materials_infile 14
@m materialsfile 15
@m pmi_infile 16
@m pmifile 17
@m crampproblemfile 18 // Not used in public version of code
@m tallyfile 19
@m outputfile 20
@m oldsourcefile 21
@m tally_infile 22
@m snapshotfile 23
@m filenames_max 23 // Largest of the above
```

[/Users/dstotler/degas2/src/readfilenames.hweb] common block definitions for filenames

```
"classes.f" 56.2 ≡
    package_init(rf)
@#if 0
    define_var_pk(rf, filenames_max, INT)
@#endif
    define_dimen_pk(rf, filenames_size, FILELEN)
    define_dimen_pk(rf, filenames_ind, filenames_max)
    define_varp_pk(rf, filenames_array, CHAR, filenames_size, filenames_ind)
    package_end(rf)
```

57 Description of file name usage in DEGAS 2

[/Users/dstotler/degas2/src/readfilenames.hweb]

The names of most input and output files used by DEGAS 2 are specified in a single file, currently called *degas2.in*. The filenames given therein are associated with the symbolic names which appear in the rest of the code. These symbolic names should be self-describing; more details about each of these files is given in the User’s Manual. The corresponding user-specified filenames (which can be an absolute or relative UNIX path name) are stored in the *filenames_array* array. The indices in that array corresponding to each symbolic name are given by the above macros. The entries in the *degas2.in* file are of the form:

`symbolic_name actual_path_name`

The entries can be in any order. Comments begin with a “#” and be on a separate line or at the end of a line. Spaces and blank lines are ignored. Unneeded filenames (see comments at the top of this file) do not have to be included. If you forget to specify a needed filename, DEGAS 2 will should give an error like:

```
Assertion failed (pmimatread.web): tempfile!=char_undef
```

An example *degas2.in* file is distributed in the *src* directory of DEGAS 2. Other examples are included in *examples* directories. The *degas2.in* file *must* be in your current “run” directory (e.g., *degas2/SUN*). Files referred to by *degas2.in* in the **data** directory should not be modified by the user. If changes need to be made, the user should make copies of the files in the “run” directory or elsewhere. The user is advised to choose filenames which will aid in the tracking of run variants.

To add new files to the above list:

1. Create a suggestive symbolic file name,
2. Assign the next integer to it in the above macro list,
3. Increment *filenames_max*,
4. Make corresponding changes to *readfilenames.web* (see the introduction there).

58 Specification for a 2-D Geometry

[/Users/dstotler/degas2/src/geometry2d.hweb]

```
$Id: fcf37c16bf459380193adee437e84bea4ee09e88 $
```

[/Users/dstotler/degas2/src/geometry2d.hweb] Parameters

```
"classes.f" 58.1 ≡
@m g2_num_points 800
@m g2_xz 2
@m g2_x 1
@m g2_z 2
```

```
[/Users/dstotler/degas2/src/geometry2d.hweb] Variable Definitions

"classes.f" 58.2 ≡
    package_init(g2)
    define_dimen_pk(g2, g2_points_ind0, 0, g2_num_points - 1)
    define_dimen_pk(g2, g2_points_ind, g2_num_points)
    define_dimen_pk(g2, g2_xz_ind, g2_xz)
    define_var_pk(g2, g2_num_polygons, INT)
    define_dimen_pk(g2, g2_poly_ind, g2_num_polygons)
    define_varp_pk(g2, g2_polygon_xz, FLOAT, g2_xz_ind, g2_points_ind0, g2_poly_ind)
    define_varp_pk(g2, g2_polygon_segment, INT, g2_points_ind0, g2_poly_ind)
    define_varp_pk(g2, g2_polygon_points, INT, g2_poly_ind)
    define_varp_pk(g2, g2_polygon_zone, INT, g2_poly_ind)
    define_varp_pk(g2, g2_polygon_stratum, INT, g2_poly_ind)
    // This is the minimum; fill out the rest later
    package_end(g2)
```

59 Two-D geometry specification class

[/Users/dstotler/degas2/src/geometry2d.hweb]

[/Users/dstotler/degas2/src/geometry2d.hweb] Define a 2-D geometry. Prefix is g2.

g2_num_points Maximum number of points in a polygon or wall. Have chosen to fix this rather than dealing with two variable dimensions.

g2_xz Just an index to represent *x* and *z*.

g2_polygon_xz_{poly_ind,points_ind,xz_ind} *x* (*xz_ind* = 1), *z* (*xz_ind* = 2) coordinates of polygon *poly_ind*. The first point (*points_ind* = 0) and last (*points_ind* = *g2_polygon_points_{poly_ind}*) must coincide. All polygons used to specify this geometry are contained in this array.

g2_polygon_segment_{poly_ind,points_ind} A label for each point and segment of a polygon. For polygons specified directly on input, just corresponds to the number of the point. For triangles generated from polygons, the original segment number is retained during the decomposition process and assigned to this array so that the edges of the new triangles can be related to the original polygon segments.

g2_polygon_points_{poly_ind} Actual number of distinct (i.e., not counting the duplicate first / last) points in polygon *poly_ind*.

g2_polygon_zone_{poly_ind} Zone number to be associated with polygon *poly_ind*.

g2_polygon_stratum_{poly_ind} Stratum number specified on input with either this polygon or its parent polygon (if decomposed into triangles).

60 Sampled origin data for flights

[/Users/dstotler/degas2/src/flight_frag.hweb]

\$Id: \$

[/Users/dstotler/degas2/src/flight_frag.hweb] Established this entity so that in parallel runs, the master can sample the initial positions and velocities of all flights, so that the source descriptions do not have to be propagated to all of the slaves. This was introduced to address a problem with the “snapshot” source in time dependent runs, in which the source data were occupying more memory than the geometry or output, leading to unacceptable memory requirements.

Subsequent testing showed, however, that this approach was not satisfactory for other runs with puff or plate sources. To this end, the *MASTER_SAMPLE* switch was introduced. Set to “no” in the Makefile.local, the source class data are broadcast to the slaves and the source particles are sampled there. Set to “yes” causes all sampling to be performed on the master; the bulk of the sources arrays are kept only on the master.

```
"classes.f" 60.1 ≡
@m ff_decls integer ff_temp

@m ff_int_number 1
@m ff_int_source 2
@m ff_int_source_kseg 3
@m ff_int_source_xseg 4
@m ff_int_source_type 5
@m ff_int_source_root_sp 6
@m ff_int_pt_sp 7
@m ff_int_pt_test 8
@m ff_int_lc_cell 9
@m ff_int_lc_zone 10
@m ff_int_lc_face 11
@m ff_int_lc_cell_next 12
@m ff_int_lc_zone_next 13
@m ff_int_lc_sector 14
@m ff_int_lc_sector_next 15
@m ff_int_pt_type 16
@m ff_int_pt_author 17
@m ff_int_max 17

@m ff_float_pt_t 1 // These are exactly the same as sn_float
@m ff_float_pt_w 2 // Can we use them instead?
@m ff_float_pt_v1 3
@m ff_float_pt_v2 4
@m ff_float_pt_v3 5
@m ff_float_lc_x1 6
@m ff_float_lc_x2 7
@m ff_float_lc_x3 8

@m ff_float_pt_max 8

@m ff_copy(i, j, is, ks, xs, type, rsp, px, rx) ff_particles_inti.ff_int_number = j
ff_particles_inti.ff_int_source = is
ff_particles_inti.ff_int_source_kseg = ks
ff_particles_inti.ff_int_source_xseg = xs
ff_particles_inti.ff_int_source_type = type
ff_particles_inti.ff_int_source_root_sp = rsp
ff_particles_inti.ff_int_pt_sp = pt_sp(px)
ff_particles_inti.ff_int_pt_test = pt_test(px)
ff_particles_inti.ff_int_lc_cell = lc_cell(pt_loc(px))
```

```

ff_particles_inti.ff_int_lc_zone = lc_zone(pt_loc(px))
ff_particles_inti.ff_int_lc_face = lc_face(pt_loc(px))
ff_particles_inti.ff_int_lc_cell_next = lc_cell_next(pt_loc(px))
ff_particles_inti.ff_int_lc_zone_next = lc_zone_next(pt_loc(px))
ff_particles_inti.ff_int_lc_sector = lc_sector(pt_loc(px))
ff_particles_inti.ff_int_lc_sector_next = lc_sector_next(pt_loc(px))
ff_particles_inti.ff_int_pt_type = pt_type(px)
ff_particles_inti.ff_int_pt_author = pt_author(px)
ff_particles_floati.ff_float_pt_t = pt_t(px)
ff_particles_floati.ff_float_pt_w = pt_w(px)
ff_particles_floati.ff_float_pt_v1 = pt_v(px)1
ff_particles_floati.ff_float_pt_v2 = pt_v(px)2
ff_particles_floati.ff_float_pt_v3 = pt_v(px)3
ff_particles_floati.ff_float_lc_x1 = lc_x(pt_loc(px))1
ff_particles_floati.ff_float_lc_x2 = lc_x(pt_loc(px))2
ff_particles_floati.ff_float_lc_x3 = lc_x(pt_loc(px))3
ff_ran_indexi = rn_index(rx)
do ff_temp = 0, ran_k - 1
  ff_ran_arrayi,ff_temp = rn_array(rx)ff_temp
end do

@m ff_label_init(j,fx,rel_wt) // Replaces fl_label and fl_init
fl_number(fx) = ff_particles_intj.ff_int_number
fl_source(fx) = ff_particles_intj.ff_int_source
fl_source_kseg(fx) = ff_particles_intj.ff_int_source_kseg
fl_source_xseg(fx) = ff_particles_intj.ff_int_source_xseg
fl_source_type(fx) = ff_particles_intj.ff_int_source_type
fl_source_root_sp(fx) = ff_particles_intj.ff_int_source_root_sp
pt_sp(fl_origin(fx)) = ff_particles_intj.ff_int_pt_sp
pt_test(fl_origin(fx)) = ff_particles_intj.ff_int_pt_test
vc_set(lc_x(pt_loc(fl_origin(fx))), ff_particles_floatj,ff_float_lc_x1, ff_particles_floatj,ff_float_lc_x2,
       ff_particles_floatj,ff_float_lc_x3)
lc_cell(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_cell
if (lc_cell(pt_loc(fl_origin(fx))) ≡ int_uninit) then
  lc_set_a(pt_loc(fl_origin(fx)))
  assert((zn_type(lc_zone(pt_loc(fl_origin(fx)))) ≡ zn_vacuum) ∨
         (zn_type(lc_zone(pt_loc(fl_origin(fx)))) ≡ zn_plasma))
else
  lc_zone(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_zone
  lc_face(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_face
  lc_cell_next(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_cell_next
  lc_zone_next(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_zone_next
  lc_sector(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_sector
  lc_sector_next(pt_loc(fl_origin(fx))) = ff_particles_intj.ff_int_lc_sector_next
end if
pt_type(fl_origin(fx)) = ff_particles_intj.ff_int_pt_type
pt_author(fl_origin(fx)) = ff_particles_intj.ff_int_pt_author

```

```

 $pt\_t(fl\_origin(fx)) = ff\_particles\_float_{j,ff\_float\_pt\_t}$ 
 $rel\_wt = ff\_particles\_float_{j,ff\_float\_pt\_w}$ 
 $pt\_w(fl\_origin(fx)) = one$ 
 $vc\_set(pt\_v(fl\_origin(fx)), ff\_particles\_float_{j,ff\_float\_pt\_v1}, ff\_particles\_float_{j,ff\_float\_pt\_v2},$ 
 $ff\_particles\_float_{j,ff\_float\_pt\_v3})$ 
 $rn\_index(fl\_rand(fx)) = ff\_ran\_index_j$ 
do  $ff\_temp = 0, ran\_k - 1$ 
 $rn\_array(fl\_rand(fx))_{ff\_temp} = ff\_ran\_array_{j,ff\_temp}$ 
end do
 $fl\_pointer(fx) = 1$ 
 $pt\_copy(fl\_origin(fx), fl\_current(fx))$ 

```

[/Users/dstotler/degas2/src/flight_frag.hweb] Variable definitions

```

"classes.f" 60.2 ≡
  package_init(ff)
    define_var_pk(ff, ff_particles_dim, INT)
    define_var_pk(ff, ff_number_particles, INT)

    define_dimen_pk(ff, ff_particles_ind, ff_particles_dim)
    define_dimen_pk(ff, ff_pt_float_ind, ff_float_pt_max)
    define_dimen_pk(ff, ff_int_ind, ff_int_max)
    define_dimen_pk(ff, ff_ran_ind, 0, ran_k - 1)

    define_varp_pk(ff, ff_particles_int, INT, ff_int_ind, ff_particles_ind)
    define_varp_pk(ff, ff_particles_float, FLOAT, ff_pt_float_ind, ff_particles_ind)
    define_varp_pk(ff, ff_ran_index, INT, ff_particles_ind)
    define_varp_pk(ff, ff_ran_array, FLOAT, ff_ran_ind, ff_particles_ind)

  package_end(ff)

```

61 Description of the flight fragment data structures

[/Users/dstotler/degas2/src/flight_frag.hweb]

[/Users/dstotler/degas2/src/flight_frag.hweb] Prefix used for these structures is *ff*.

ff_particles_dim Explicit variable used to track the size of the arrays.

ff_number_particles Number of particles contained in the fragment.

ff_particles_int Integer parameters associated with the particles.

ff_particles_float Floating point parameters associated with the particles.

ff_ran_index Index into the random number sequence for each particle.

ff_ran_array Random number sequence for each particle.

62 INDEX

`_base`: 30.6, 37.5, 41.6, 45.1, 45.3, 47.1, 47.3.
`_base_1`: 37.5, 41.6.
`_delta`: 45.1, 47.1.
`_dep_var_dim`: 30.6.
`_eval_name`: 45.1, 47.1.
`_grp`: 33.1.
`_id`: 52.2.
`_inc`: 37.5, 41.6.
`_m`: 50.2, 50.5.
`_min`: 45.1, 47.1.
`_mod`: 50.7.
`_paste`: 50.1.
`_rank`: 45.1, 47.1.
`_register`: 50.6.
`_size`: 30.6, 37.5, 41.6, 45.3, 47.3.
`_spacing`: 45.1, 47.1.
`_tab`: 37.5, 41.6, 45.1, 45.3, 47.1, 47.3.
`_tab_index`: 45.1, 47.1.
`_var`: 45.1, 47.1.
`_1`: 30.6, 37.5, 41.6, 45.3, 47.3, 50.3.
`_2`: 37.5, 41.6.

`aladinfile`: 56.1.
`aladoutfile`: 56.1.
`alb`: 26.3.
`albedo`: 27.1.
`array`: 1, 51.
`array_arglist`: 50.2, 50.4.
`array_arglistc`: 50.2, 50.4.
`array_rank`: 50.2.
`array_shape`: 50.2.
`array_shape_star`: 50.2.
`array_shape_stara`: 50.2.
`array_size`: 45.3, 47.3, 50.2, 50.4, 50.5, 51.
`array_sizea`: 50.5.
`array_sizeb`: 50.5.
`array_sizec`: 50.5.
`array_sizev`: 50.2, 50.4, 51.
`array_unit`: 50.2, 50.5, 51.
`arrayname`: 51.
`assert`: 20.2, 20.3, 52.2, 60.1.
`ASSIGN`: 50.4.
`atomic_mass_unit`: 39.1.
`atomic_mass_unit_g`: 39.1.
`author_`: 8.1.

`back`: 43.3, 44.3.
`back_ind`: 33.1.
`background`: 21.1.
`background_coords`: 20.4, 21.1.

`background_ind`: 20.4.
`background_n`: 20.1, 20.4.
`background_temp`: 20.1, 20.4.
`background_v`: 20.1, 20.4.
`backgroundfile`: 56.1.
`base`: 23.4.
`bk`: 20.4, 21.1.
`bk_args`: 20.1.
`bk_check`: 20.1.
`bk_copy`: 20.1.
bk_decl: 20.1.
`bk_dummy`: 20.1.
`bk_ms`: 20.1.
`bk_mx`: 20.1.
`bk_my`: 20.1.
`bk_n`: 20.1, 21.1.
`bk_num`: 20.1, 20.4.
`bk_plasma_ind`: 20.4.
`bk_temp`: 20.1, 21.1.
`bk_v`: 20.1, 21.1.
`bk_xpt`: 20.1.
`bool_list`: 39.4.
`bool_sect`: 39.4.
`boxgen`: 23.3.

`cat`: 50.6, 50.7.
`cell_`: 10.
`cell_exit`: 8.2.
`cell_next_`: 10.
`CHAR`: 4.3, 6.3, 14.3, 16.3, 18.2, 22.3, 24.3, 26.2,
 28.3, 30.4, 32.3, 37.3, 39.2, 41.4, 43.4, 45.2, 47.2,
 50.1, 50.2, 51, 54.3, 56.2.
`char_alloc`: 50.4.
`char_free`: 50.4.
`char_mem`: 50.4.
`char_realloc`: 50.4.
`char_spec`: 50.2.
`check_location`: 10.
`check_tally`: 30.1, 30.5.
`check_zone`: 24.1.
`CL`: 51.
`CL_args`: 1.
`CL_check`: 1.
`CL_common`: 1, 51.
`CL_common_a`: 51.
`CL_common_b`: 51.
`CL_copy`: 1.
`CL_decl`: 1.
`CL_dummy`: 1.
`CL_ndecl`: 51.
`CL_ncdef`: 51.
`CL_ncread`: 51.
`CL_ncreada`: 51.

CL_ncwrite: 51.
CL_pvmget: 51.
CL_pvmput: 51.
COMMA: 50.1.
common: 39.3.
const: 10.3, 22.1, 22.2, 39.1.
coord: 20.2, 20.3.
COUNT_: 50.6.
cpl_ind: 33.1.
cramdproblemfile: 56.1.
cv_ind: 31.1.

d_sector: 27.2.
data_base: 23.4.
data_dim: 23.4.
data_filename: 39.2.
data_size: 23.4.
de: 28.3, 29.1.
de_algorithm_circular: 28.2.
de_algorithm_uniform: 28.2, 29.1.
de_algorithm_unknown: 28.2.
de_args: 28.1.
de_decl: 28.1.
de_dummy: 28.1.
de_grp_ind: 28.3.
de_grps: 28.1, 28.3, 29.1.
de_lookup: 28.1, 29.2.
de_max_bins: 27.1, 28.3, 29.1.
de_name_len: 28.2, 28.3.
de_name_string: 28.3.
de_spacing_linear: 28.2.
de_spacing_log: 28.2.
de_spacing_unknown: 28.2.
de_start_end_ind: 28.3.
de_sy_len: 28.2, 28.3.
de_symbol_string: 28.3.
de_tot_view_ind: 28.3.
de_uniform_circular: 29.1.
de_var_unknown: 28.2.
de_var_wavelength: 28.2, 29.1.
de_view_algorithm: 28.3, 29.1.
de_view_base: 28.2, 28.3, 29.1.
de_view_end: 28.2, 28.3, 29.1.
de_view_halfwidth: 28.3, 29.1.
de_view_ind: 28.3.
de_view_pointer: 28.2, 29.2.
de_view_points: 28.3, 29.1.
de_view_size: 28.3, 29.1.
de_view_start: 28.2, 28.3, 29.1.
de_view_tab: 28.2, 28.3, 29.1.
de_zone_frags: 28.3, 29.1.
de_zone_frags_dim: 28.3, 29.1.
de_zone_frags_ind: 28.3.

de_zone_frags_max_zn: 28.3, 29.1.
de_zone_frags_min_zn: 28.3, 29.1.
de_zone_frags_num: 28.3, 29.1.
de_zone_frags_size: 28.3, 29.1.
de_zone_frags_start: 28.3, 29.1.
de_zone_frags_zone: 29.1.
de_zone_frags_zones: 28.3, 29.1.
decl_alpcom: 39.3.
declare_var: 50.2.
declare_varp: 50.2.
define_dimen: 50.2.
define_dimen_pk: 4.3, 6.3, 14.3, 16.3, 18.2, 20.4,
 22.3, 24.3, 26.2, 28.3, 30.4, 32.3, 34.2, 37.3, 39.2,
 41.4, 43.4, 43.5, 45.2, 47.2, 50.6, 51, 54.3, 56.2,
 58.2, 60.2.
define_dummy_pk: 50.6.
define_sector_exit: 26.3, 27.2.
define_sector_plasma: 26.3, 27.2.
define_sector_target: 26.3, 27.2.
define_sector_vacuum: 26.3, 27.2.
define_sector_wall: 26.3, 27.2.
define_tally: 31.1.
define_var: 50.2.
define_var_pk: 1, 4.3, 6.3, 14.3, 16.3, 18.2, 20.4,
 22.3, 24.3, 26.2, 28.3, 30.4, 32.3, 34.2, 37.3, 39.2,
 41.4, 43.4, 43.5, 45.2, 47.2, 50.6, 51, 54.3, 56.2,
 58.2, 60.2.
define_varlocal_pk: 4.3, 6.3, 14.3, 16.3, 18.2, 22.3,
 24.3, 30.4, 32.3, 37.3, 41.4, 43.4, 47.2, 50.6.
define_varp: 34.1, 50.2, 50.6.
define_varp_pk: 1, 4.3, 6.3, 14.3, 16.3, 18.2, 20.4,
 22.3, 24.3, 26.2, 28.3, 30.4, 32.3, 34.2, 37.3, 41.4,
 43.4, 43.5, 45.2, 47.2, 50.6, 51, 54.3, 56.2, 58.2,
 60.2.
degas2: 57.
dep_var_ind: 37.3, 38.1.
dest: 50.7.
detector: 27.1.
detector_delta: 28.3, 29.1.
detector_min: 28.3, 29.1.
detector_name: 28.1, 28.3, 29.1.
detector_num_views: 28.3, 29.1, 29.2.
detector_spacing: 28.3, 29.1.
detector_tab_index: 28.3, 29.1.
detector_total_views: 28.3, 29.1, 29.2.
detector_var: 28.3, 29.1.
detector_view_base: 29.1.
detector_view_setup: 29.1.
diag_grp_ind: 26.2.
diag_lookup: 26.1, 27.2.
diagnostic: 26.2, 27.1.
diagnostic_delta: 26.2, 27.1.

diagnostic_grp_base: 26.1, 26.2, 27.1.
diagnostic_grp_name: 26.1, 26.2, 27.1.
diagnostic_ind: 27.1.
diagnostic_min: 26.2, 27.1.
diagnostic_num_sectors: 26.1, 26.2, 27.1.
diagnostic_sector: 26.1, 27.1, 27.2.
diagnostic_sector_tab: 26.1, 26.2, 27.1.
diagnostic_spacing: 26.2, 27.1.
diagnostic_tab_index: 26.2, 27.1.
diagnostic_var: 26.2, 27.1.
dim_junk_array: 50.2.
dimen_def: 52.2, 53.
dimen_inq: 52.2, 53.
dimid: 52.2, 53.
dimid1: 52.2.
dimname: 51, 53.
dimname1: 51.
dimname2: 51.
dims: 50.3.
do_common: 50.7.
do_module: 50.7.
do_mpibcast: 50.7.
do_mpibcastna: 50.7.
do_mpireceive: 50.7.
do_mpisend: 50.7.
do_ncdecl: 50.7.
do_ncdef: 50.7.
do_ncread: 50.7.
do_ncwrite: 50.7.
do_netcdfa: 50.7.
do_pvmget: 50.7.
do_pvmpput: 50.7.

e_bins: 22.1, 23.1.
e_sector: 27.2.
el: 4.3, 5.1.
el_args: 4.1.
el_check: 4.1.
el_copy: 4.1.
el_decl: 4.1.
el_dummy: 4.1.
el_lookup: 4.1, 5.2.
el_m: 4.1, 5.1.
el_name: 1, 4.1, 5.1.
el_name_len: 4.2, 4.3, 5.1.
el_num: 4.1, 4.3.
el_sy: 4.1, 5.1.
el_sy_len: 4.2, 4.3, 5.1.
el_z: 4.1, 5.1.
electron_charge: 39.1.
element: 5.1.
element_ind: 4.3.
element_m: 4.1, 4.3.

element_name: 1, 4.1, 4.3.
element_name_string: 4.3.
element_sy: 4.1, 4.3.
element_symbol_string: 4.3.
element_version: 4.3.
element_z: 4.1, 4.3.
elements_infile: 56.1.
elementsfile: 56.1.
emission_rate_h_alpha: 31.1.
end: 50.7.
end_common: 50.7.
end_dep_var: 39.4.
end_indep_var: 39.4.
end_module: 50.7.
end_mpibcast: 50.7.
end_mpibcastna: 50.7.
end_mpireceive: 50.7.
end_mpisend: 50.7.
end_ncdecl: 50.7.
end_ncdef: 50.7.
end_ncread: 50.7.
end_ncwrite: 50.7.
end_pvmget: 50.7.
end_pvmpput: 50.7.
energy: 31.1.
energy_change: 31.1.
epsilon: 39.1.
epsilon_angle: 10.3.
equivalents: 43.5, 44.4.
est_ind: 31.1.
ev_to_ergs: 39.1.
eval_name: 37.3.
examples: 57.
exit: 26.2, 27.1.
exit_albedo: 26.2, 26.3.
exit_ind: 26.2, 27.1.
exit_sector: 26.2, 26.3, 27.1.

face: 26.1, 27.2.
FALSE: 22.2, 23.3, 33.1, 35.1.
ff: 60.2, 61.1.
ff_copy: 60.1.
ff_decls: 60.1.
ff_float_lc_x1: 60.1.
ff_float_lc_x2: 60.1.
ff_float_lc_x3: 60.1.
ff_float_pt_max: 60.1, 60.2.
ff_float_pt_t: 60.1.
ff_float_pt_v1: 60.1.
ff_float_pt_v2: 60.1.
ff_float_pt_v3: 60.1.
ff_float_pt_w: 60.1.
ff_int_ind: 60.2.

ff_int_lc_cell: 60.1.
ff_int_lc_cell_next: 60.1.
ff_int_lc_face: 60.1.
ff_int_lc_sector: 60.1.
ff_int_lc_sector_next: 60.1.
ff_int_lc_zone: 60.1.
ff_int_lc_zone_next: 60.1.
ff_int_max: 60.1, 60.2.
ff_int_number: 60.1.
ff_int_pt_author: 60.1.
ff_int_pt_sp: 60.1.
ff_int_pt_test: 60.1.
ff_int_pt_type: 60.1.
ff_int_source: 60.1.
ff_int_source_kseg: 60.1.
ff_int_source_root_sp: 60.1.
ff_int_source_type: 60.1.
ff_int_source_xseg: 60.1.
ff_label_init: 60.1.
ff_number_particles: 60.2, 61.1.
ff_particles_dim: 60.2, 61.1.
ff_particles_float: 60.1, 60.2, 61.1.
ff_particles_ind: 60.2.
ff_particles_int: 60.1, 60.2, 61.1.
ff_pt_float_ind: 60.2.
ff_ran_array: 60.1, 60.2, 61.1.
ff_ran_ind: 60.2.
ff_ran_index: 60.1, 60.2, 61.1.
ff_temp: 60.1.
fff: 35.1, 35.2.
fileid: 50.7, 52.2, 53.
FILELEN: 14.3, 18.2, 56.2.
filenames_array: 56.2, 57.
filenames_ind: 56.2.
filenames_max: 56.1, 56.2, 57.
filenames_size: 56.2.
fin: 34.2, 35.1.
find_threshold: 39.1.
fit: 38.1.
fit_coef: 39.4.
fl: 13.1.
fl_args: 12.1.
fl_check: 12.1.
fl_common: 12.1.
fl_copy: 12.1.
fl_current: 1, 12.1, 12.3, 13.1, 60.1.
fl_decl: 12.1.
fl_dummy: 12.1.
fl_init: 12.3, 13.2, 60.1.
fl_label: 12.3, 13.2, 60.1.
fl_number: 12.1, 12.3, 13.1, 13.2, 60.1.
fl_origin: 12.1, 12.3, 13.1, 13.2, 60.1.
fl_pointer: 12.1, 12.3, 13.1, 60.1.
fl_rand: 12.1, 13.1, 60.1.
fl_source: 12.1, 12.3, 13.1, 13.2, 60.1.
fl_source_kseg: 12.1, 12.3, 13.1, 60.1.
fl_source_root_sp: 12.1, 12.3, 13.1, 60.1.
fl_source_type: 12.1, 12.3, 13.1, 60.1.
fl_source_xseg: 12.1, 12.3, 13.1, 60.1.
fl_stack: 12.1, 13.1.
fl_stack_max: 12.1, 12.2, 13.1.
fl_temp: 12.1.
flight: 13.1, 13.2.
flighttest: 23.1.
FLOAT: 4.3, 6.3, 20.4, 22.3, 24.3, 26.2, 28.3, 30.4,
 32.3, 34.2, 37.3, 39.2, 41.4, 43.4, 45.2, 47.2, 50.1,
 50.2, 51, 54.3, 58.2, 60.2.
flt: 34.2, 35.1.
foo: 51.
FORTRAN77: 50.2, 50.4, 50.5, 50.7.
FORTRAN90: 50.2, 50.7.
frag: 34.2, 35.1.
fx: 60.1.
generics: 43.5, 44.4.
geom_epsilon: 10.3.
geom_infinity: 10.3.
geom_large: 10.3.
geometry: 27.1.
geometry_symmetry_cyl_hw: 20.2, 20.3.
geometry_symmetry_cyl_section: 20.2, 20.3.
geometry_symmetry_cylindrical: 20.2, 20.3.
geometry_symmetry_none: 20.2, 20.3.
geometryfile: 56.1.
geomint: 27.1.
giparameters: 23.4.
giparams: 23.4.
gparameters: 23.4.
gparams: 23.4.
groud_ind: 27.1.
group: 27.2.
group_ind: 27.1, 29.1, 29.2, 33.1.
grp: 26.1, 27.2, 28.2.
grp_ind: 33.1.
g2: 58.2.
g2_num_points: 58.1, 58.2, 59.1.
g2_num_polygons: 58.2.
g2_points_ind: 58.2.
g2_points_ind0: 58.2.
g2_poly_ind: 58.2.
g2_polygon_points: 58.2, 59.1.
g2_polygon_segment: 58.2, 59.1.
g2_polygon_stratum: 58.2, 59.1.
g2_polygon_xz: 58.2, 59.1.
g2_polygon_zone: 58.2, 59.1.

g2_x: [58.1](#).
g2_xz: [58.1](#), 58.2, 59.1.
g2_xz_ind: 58.2.
g2_z: [58.1](#).
half: 22.1.
hweb: 1, 27.1, 30.6, 31.1, 38.2, 42.1, 42.2, 44.4, 45.3, 46.1, 48.1.
i_float: 55.1.
i_int: 55.1.
IBM: 50.2.
igroup: 23.1, 23.4.
implicit_none_f90mod: 50.7.
in: 57.
inc: [30.5](#), 30.6, 37.5, 41.6, [45.3](#).
include: 39.3.
ind: 32.2, 33.2.
ind_: 30.6, 37.5, 41.6, 45.3, 47.3, 50.2, 50.3, 50.5.
ind_temp_m: 50.2.
ind_temp_1: 50.2.
ind_temp_2: 50.2.
ind_temp_3: 50.2.
ind_x_m: 50.2.
ind_x_n: 50.2.
independent_paramaters: 41.2.
index_parameters: 31.2.
indx_m: 50.5.
init_base: 45.3, 47.3.
init_var0_list: 31.1, 43.3.
inst: 1.
inst1: 1.
inst2: 1.
int: 50.5.
INT: 4.3, 6.3, 14.3, 16.3, 18.2, 20.4, 22.3, 24.3, 26.2, 28.3, 30.4, 32.3, 34.2, 37.3, 39.2, 41.4, 43.4, 43.5, 45.2, 47.2, [50.1](#), 51, 54.3, 56.2, 58.2, 60.2.
int_alloc: [50.4](#).
int_free: [50.4](#).
int_lookup: 26.1, 43.1, [49](#).
int_mem: 50.4.
int_realloc: [50.4](#).
int_uninit: 60.1.
int_unused: 23.2.
integ_transform_exp: [39.1](#).
integ_transform_ratio: [39.1](#).
integer: 4.1, 6.1, 8.1, 10, 12.1, 14.1, 16.1, 18, 20.1, 22.1, 24.1, 26.1, 28.1, 30.1, 30.5, 36.1, 37.4, 39.4, 41.5, 43.1, 45.3, 49, 50.2, 50.4, 52.2.
integration_acc: [39.1](#).
integration_infinity: [39.1](#).
ip: 30.3, 31.2, 55.1.
iparam: 23.4.
iparameters: 23.4.
iparams: 23.4.
is: 60.1.
iseg: 23.2, 23.4.
isource: 12.3, 13.2.
iterate_pk: [50.7](#).
itype: 23.2.
ix: 33.1.
iy: 33.1.
junk: 50.2.
junk_array: 50.2.
junk_arrray: 50.2.
ks: 60.1.
kseg: 12.3, 13.2.
label_num_ind: 39.2.
last_sizea: [50.5](#).
last_sizeb: [50.5](#).
last_sizec: [50.5](#).
last_ubound: [50.5](#).
last_ubounda: [50.5](#).
last_uboundb: [50.5](#).
last_uboundc: [50.5](#).
lc: 11.1.
lc_args: 8.1, [10](#).
lc_cell: [10](#), 10.1, 10.2, 11.1, 60.1.
lc_cell_next: [10](#), 10.1, 10.2, 11.1, 60.1.
lc_cell_next1: [10](#).
lc_cell1: [10](#).
lc_check: 8.1, [10](#).
lc_copy: 8.1, [10](#).
lc_decl: 8.1, [10](#).
lc_decls: [10](#).
lc_dummy: 8.1, [10](#).
lc_face: 8.3, [10](#), 10.1, 10.2, 11.1, 60.1.
lc_face1: [10](#).
lc_sector: [10](#), 10.1, 10.2, 11.1, 60.1.
lc_sector_next: [10](#), 10.1, 10.2, 11.1, 60.1.
lc_sector_next1: [10](#).
lc_sector1: [10](#).
lc_set: [10.1](#), 11.2.
lc_set_a: [10.1](#), 60.1.
lc_set_b: [10.2](#).
lc_thru_face: 8.3, [10.2](#), 11.2.
lc_x: 8.3, [10](#), 10.1, 11.1, 60.1.
lc_x1: [10](#).
lc_zone: [10](#), 10.1, 10.2, 11.1, 60.1.
lc_zone_next: [10](#), 10.1, 10.2, 11.1, 60.1.
lc_zone_next1: [10](#).
lc_zone1: [10](#).
lcptr: [39.3](#).
lcmpptr: [39.3](#).

len: 50.2.
len_junk: 50.2.
len_junk_array: 50.2.
leseqn: 39.3.
leval: 39.4.
list: 23.4.
list_dim: 23.4.
list_size: 23.4.
locate_point: 10, 10.1.
location: 11.1, 11.2.
lower_junk_array: 50.2.
ma: 16.3, 17.1, 19.1, 27.1, 27.2, 44.1.
ma_args: 16.1.
ma_check: 16.1.
ma_copy: 16.1.
ma_decl: 16.1.
ma_dummy: 16.1.
ma_lookup: 16.1, 17.3.
ma_name: 16.1, 17.1.
ma_name_len: 16.2, 16.3, 17.1.
ma_num: 16.1, 16.3, 17.2, 43.4.
ma_sy: 16.1, 17.1.
ma_sy_len: 16.2, 16.3, 17.1.
macro_index: 42.1, 46.1, 48.1.
mass: 31.1.
mass_change: 31.1.
mass_unit_string: 39.2.
MASTER_SAMPLE: 60.1.
mat: 26.3.
material: 17.1, 27.2.
materials_ind: 16.3.
materials_infile: 56.1.
materials_name: 16.1, 16.3.
materials_name_string: 16.3.
materials_sy: 16.1, 16.3.
materials_symbol_string: 16.3.
materials_version: 16.3.
materialsfile: 56.1.
max: 50.2, 51.
max_: 50.2.
max_bins: 27.1, 29.1, 30.3.
MAX_DIM: 50.1, 50.2, 52.2.
max_integ_steps: 39.1.
max_interp_pts: 39.1.
max_name: 50.2.
MAX_NC_NAME: 52.1, 52.2.
mem_alloc: 50.4.
mem_alloc_: 50.4.
mem_assign: 50.4.
mem_free: 50.4.
mem_free_: 50.4.
mem_inc: 50.5, 51.
mem_realloc: 50.4.
mem_realloc_: 50.4.
mem_size: 50.5.
min: 50.2, 51.
min_: 50.2.
min_bound_ratio: 39.1.
min_name: 50.2.
mod: 50.5.
module: 50.7.
mom_ind: 33.1.
moment_number: 39.4.
momentum_change_vector: 31.1.
momentum_vector: 31.1.
MPI: 50.7.
msg_length: 39.1.
mult: 48.1.
name: 26.1, 27.2, 28.1, 29.2, 50.2, 50.6, 52.2, 53.
nbl: 39.3.
nblmx: 39.3.
nc_attr: 52.2.
NC_BYTE: 52.1.
NC_CHAR: 52.1.
NC_CLOBBER: 52.1.
nc_corner: 52.2, 53.
nc_decls: 52.2.
nc_dims: 52.2, 53.
NC_DOUBLE: 52.1.
nc_dummy: 52.2.
nc_edge: 52.2, 53.
NC_FATAL: 52.1.
NC_FLOAT: 52.1.
NC_GLOBAL: 52.1.
NC_INT: 52.1.
NC_LONG: 52.1.
NC_NOCLOBBER: 52.1.
NC_NOWRITE: 52.1.
nc_rank: 52.2.
NC_REAL: 52.1.
NC_SHORT: 52.1.
nc_size: 52.2.
nc_stat: 52.2.
nc_type: 52.2.
NC_VERBOSE: 52.1.
NC_WRITE: 52.1.
ncagt: 52.1.
ncagtc: 52.1.
ncapt: 52.1.
ncaptc: 52.1.
ncattget: 52.1.
ncattgetc: 52.1.
ncattput: 52.1.
ncattputc: 52.1.

nclos: 52.1.
ncclose: 52.1.
nccre: 52.1.
nccreate: 52.1, 52.2.
ncddef: 52.1.
ncdid: 52.1.
ncdimdef: 52.1, 52.2, 53.
ncdimid: 52.1, 52.2.
ncdiminq: 52.1, 52.2, 53.
ncding: 52.1.
ncendef: 52.1.
ncendf: 52.1.
ncfln: 39.3.
ncmln: 39.3.
nconversions: 30.4, 31.1.
ncopen: 52.1, 52.2.
ncopn: 52.1.
ncedef: 52.1.
ncrdef: 52.1.
ncvardef: 52.1, 52.2, 53.
ncvarget: 52.1, 53.
ncvargetc: 52.1, 53.
ncvarid: 52.1, 52.2.
ncvarinq: 52.1, 52.2.
ncvarput: 52.1, 53.
ncvarputc: 52.1, 53.
ncvdef: 52.1.
ncvgt: 52.1.
ncvgtc: 52.1.
ncvid: 52.1.
ncving: 52.1.
ncvpt: 52.1.
ncvptc: 52.1.
neln: 39.3.
nelmx: 39.3.
NETCDFV2: 52.1.
new_bool_sect: 39.4.
newmax: 51.
next_bool: 39.4.
next_dep_var: 39.4.
next_hlab: 39.4.
next_line: 39.4.
next_token: 49.
nf_get_vara_text: 52.2, 53.
nf_inq_format: 52.2.
NF_NETCDF4: 52.1.
NF_64BIT_OFFSET: 52.1.
nhl: 39.3.
nhlmx: 39.3.
nsectors: 26.1, 26.2, 27.1.
num: 12.3, 13.2, 23.4.
num_equiv: 43.5, 44.4.
num_evaluations: 39.4.
num_extrap_pts: 39.1.
num_fit_coef: 39.4.
num_generics: 43.5, 44.4.
number_: 12.1.
o_mean: 32.2, 32.3, 33.1, 34.2.
o_var: 32.2, 32.3, 33.1, 34.2.
old_size: 37.4, 37.5, 41.5, 41.6.
oldmax: 51.
oldsourcefile: 56.1.
on: 50.4, 50.5.
one: 8.3, 22.1, 60.1.
origin_: 12.1.
ou: 32.3, 33.1.
ou_back_ind: 32.3.
ou_back_max: 32.2.
ou_coupling_energy: 32.2.
ou_coupling_mass: 32.2.
ou_coupling_max: 32.2, 32.3.
ou_coupling_momentum_1: 32.2.
ou_coupling_momentum_2: 32.2.
ou_coupling_momentum_3: 32.2.
out_array_index: 32.2, 33.2.
out_index_5: 32.2.
out_post: 33.1.
out_post_all: 32.3, 33.1.
out_post_grp: 32.3.
output_all: 32.3, 33.1.
output_checkpoint: 32.3, 33.1.
output_coupling_ind: 32.3.
output_grp: 1, 32.3, 33.1.
output_grps_ind: 32.3.
output_ind_1: 32.3.
output_ind_2: 32.3.
output_index_1_max: 32.3, 33.1.
output_index_1_min: 32.3, 33.1.
output_index_2_max: 32.3, 33.1.
output_index_2_min: 32.3, 33.1.
output_moments_ind: 32.3.
output_num_flights: 32.3, 33.1.
output_old_file: 32.3.
output_random_seed: 32.3, 33.1.
output_seed_ind: 32.3.
output_tab_ind: 32.3.
output_version: 32.3.
output_weight_grp: 32.3, 33.1.
output_2D_coupling: 32.3, 33.1.
outputfile: 56.1.
p_sector: 27.2.

package_end: 4.3, 6.3, 14.3, 16.3, 18.2, 20.4, 22.3, 24.3, 26.2, 28.3, 30.4, 32.3, 34.2, 37.3, 39.2, 41.4, 43.4, 43.5, 45.2, 47.2, 50.6, 54.3, 56.2, 58.2, 60.2.
package_init: 4.3, 6.3, 14.3, 16.3, 18.2, 20.4, 22.3, 24.3, 26.2, 28.3, 30.4, 32.3, 34.2, 37.3, 39.2, 41.4, 43.4, 43.5, 45.2, 47.2, 50.6, 51, 54.3, 56.2, 58.2, 60.2.
parameters: 23.4.
params: 23.4.
parse_string: 49.
part_ind: 31.1.
particle: 9.1, 9.2.
particle_track: 8.1, 8.3.
paste: 50.1, 50.2, 50.4.
pd: 42.1, 47.2, 48.1.
pd_data: 48.2.
pd_data_args: 47.1, 48.2.
pd_data_base: 48.2.
pd_data_size: 48.2.
pd_data_tab: 48.2.
pd_dep_var_ind: 47.2.
pd_dep_var_max: 47.1, 47.2.
pd_eval_string: 47.2.
pd_handling: 47.1, 48.1, 48.2.
pd_handling_ind: 47.2.
pd_inc: 47.2, 47.3.
pd_ragged_alloc: 47.3, 48.1, 48.2.
pd_ragged_realloc: 47.3, 48.1, 48.2.
pd_rank_ind: 47.2.
pd_rank_ind0: 47.2.
pd_set_baseinc: 47.3.
pd_tab_index: 48.2.
pd_tag_string: 47.2.
pd_yield: 47.1, 48.1, 48.2.
pd_yield_ind: 47.2.
pf: 19.1, 41.4, 42.1, 48.1.
pf_data: 41.6, 42.2.
pf_data_base: 41.3, 41.4, 42.1.
pf_data_inc: 41.4, 42.1.
pf_data_ind: 41.4.
pf_data_size: 41.4, 42.1.
pf_data_tab: 41.3, 41.4, 42.1, 42.2.
pf_data_table: 41.3, 42.1, 42.2.
pf_decls: 41.5.
pf_dep_var_ind: 41.4.
pf_dep_var_max: 41.2, 41.4, 47.1.
pf_eval_name: 41.4, 42.1.
pf_eval_string: 41.4.
pf_eval_string_length: 41.2, 41.4, 42.1, 47.2, 48.1.
pf_max: 41.4, 42.1.
pf_max_indep_params: 41.2.
pf_max_random: 41.2.
pf_min: 41.4, 42.1.
pf_mult: 41.4, 42.1.
pf_name: 41.4, 42.1.
pf_num_dep_var: 41.4, 42.1.
pf_ragged_alloc: 41.6, 42.1, 42.2.
pf_ragged_realloc: 41.6, 42.1, 42.2.
pf_rank: 41.4, 42.1.
pf_rank_ind: 41.4.
pf_rank_ind0: 41.4.
pf_spacing: 41.4, 42.1.
pf_spacing_linear: 41.2, 48.1.
pf_spacing_log: 41.2, 48.1.
pf_spacing_unknown: 41.2.
pf_symbol_string: 41.4.
pf_tab_index: 41.3, 41.4, 41.6, 42.1, 42.2.
pf_table_rank_max: 41.2, 41.4, 47.2.
pf_tag_string: 41.4.
pf_tag_string_length: 41.2, 41.4, 42.1, 47.2, 48.1.
pf_unit_string: 41.4.
pf_unit_string_length: 41.2, 41.4.
pf_units: 41.4, 42.1.
pf_var: 41.4, 42.1.
pf_var_cos_polar_angle_in: 41.2.
pf_var_energy_in: 41.2.
pf_var_polar_angle_in: 41.2.
pf_var_t_wall: 41.2.
pf_var_unknown: 41.2.
pf_var_vel_1_in: 41.2.
pf_var_vel_2_in: 41.2.
pf_var_vel_3_in: 41.2.
pf_var_1st_random_number: 41.2.
pf_var_2nd_random_number: 41.2.
pf_var_3rd_random_number: 41.2.
pf_version: 41.4.
pk: 50.6, 50.7, 51.
PK_CAT_MAX: 50.6, 50.7.
PK_CAT_MIN: 50.6, 50.7.
PK_DIMEN: 50.6.
PK_DUMMY: 50.6.
pk_ncdecl: 51.
PK_VAR: 50.6.
PK_VARLOCAL: 50.6.
PK_VARP: 50.6.
plasma: 26.2, 27.1.
plasma_coords_cartesian: 20.1, 21.1.
plasma_coords_cylindrical: 20.1, 20.2, 20.3, 21.1.
plasma_e_ion: 27.1.
plasma_e_ion_delta: 27.1.
plasma_e_ion_mult: 27.1.
plasma_e_ion_sheath: 27.1.
plasma.ind: 26.2, 27.1.
plasma_lookup: 26.1, 27.2.

plasma_sector: 26.1, 26.2, 26.3, 27.1.
plate: 42.1, 48.1.
plt_e_bins: 23.1.
pm: 9.2, 18.2, 19.1, 42.1, 44.1.
pm_args: 18.
pm_check: 18.
pm_copy: 18.
pm_decl: 18.
pm_dummy: 18.
pm_filename: 18, 19.1.
pm_gen: 18, 19.1.
pm_ignore: 18.2.
pm_lookup: 18, 19.3.
pm_materials: 18, 19.1.
pm_name: 18, 19.1.
pm_name_len: 18.1, 18.2, 19.1.
pm_num: 18, 18.2, 19.2, 43.4.
pm_pmi_type: 18, 19.1.
pm_product: 18, 19.1.
pm_product_max: 18.1, 18.2.
pm_product_num: 18, 19.1.
pm_reagent: 18, 19.1.
pm_sy: 18, 19.1, 42.1.
pm_sy_len: 18.1, 18.2, 19.1, 41.4, 42.1.
pm_type_len: 18.1, 18.2, 19.1.
pmi: 8.3, 19.1.
pmi_bound_product: 18.2.
pmi_bound_reagent: 18.2.
pmi_filename: 18, 18.2.
pmi_filename_string: 18.2.
pmi_generic: 18, 18.2.
pmi_generic_no: 18, 19.1.
pmi_generic_yes: 18, 19.1.
pmi_handling: 48.1, 48.2.
pmi_handling_base: 47.1, 47.2, 48.1.
pmi_handling_delta: 47.2, 48.1.
pmi_handling_eval_name: 47.2, 48.1.
pmi_handling_min: 47.2, 48.1.
pmi_handling_num_rand: 47.2, 48.1.
pmi_handling_rank: 47.2, 48.1.
pmi_handling_size: 47.2, 48.1.
pmi_handling_spacing: 47.2, 48.1.
pmi_handling_tab: 47.1, 47.2, 48.1.
pmi_handling_tab_index: 47.1, 47.2, 48.1, 48.2.
pmi_handling_var: 47.2, 48.1.
pmi_handling_var0: 47.2, 48.1.
pmi_ind: 18.2.
pmi_infile: 56.1.
pmi_materials: 18, 18.2.
pmi_materials_string: 18.2.
pmi_name: 18, 18.2.
pmi_name_string: 18.2.
pmi_product: 18, 18.2.
pmi_product_ind: 18.2.
pmi_product_num: 18, 18.2.
pmi_reagent: 18, 18.2.
pmi_sy: 18, 18.2.
pmi_symbol_string: 18.2.
pmi_type: 18, 18.2.
pmi_type_string: 18.2.
pmi_version: 18.2.
pmi_yield: 48.2.
pmi_yield_base: 47.1, 47.2, 48.1.
pmi_yield_delta: 47.2, 48.1.
pmi_yield_eval_name: 47.2, 48.1.
pmi_yield_min: 47.2, 48.1.
pmi_yield_num_rand: 47.2, 48.1.
pmi_yield_rank: 47.2, 48.1.
pmi_yield_size: 47.2, 48.1.
pmi_yield_spacing: 47.2, 48.1.
pmi_yield_tab: 47.1, 47.2, 48.1.
pmi_yield_tab_index: 47.1, 47.2, 48.1, 48.2.
pmi_yield_var: 47.2, 48.1.
pmidata: 30.6, 45.3, 46.1, 48.1.
pmifile: 56.1.
pmiformat: 38.2, 42.1, 42.2, 48.1.
pointer: 27.1, 50.2, 50.4.
pointer_: 12.1.
points_ind: 59.1.
poly_ind: 59.1.
pos_: 10.
pr: 43.4, 44.1.
pr_arrangement_max: 43.2, 43.4.
pr_background: 43.1, 44.1.
pr_background_args: 43.1.
pr_background_check: 43.1.
pr_background_copy: 43.1.
pr_background_decl: 43.1.
pr_background_dummy: 43.1.
pr_background_lookup: 43.1, 44.3.
pr_background_num: 32.2, 32.3, 43.1, 43.3, 43.4,
 44.2, 44.3.
pr_bk_rc: 43.1, 44.1.
pr_bkrc_dim: 43.4, 44.2.
pr_bkrc_num: 43.4, 44.2.
pr_bkrc_prod: 43.1, 44.1.
pr_bkrc_reagent_max: 43.2, 43.4, 44.1.
pr_bkrc_rg: 43.1, 44.1.
pr_equiv_ind: 43.5.
pr_ex_rc: 43.1, 44.1.
pr_ex_rc_num: 44.1.
pr_ex_rc_prod: 43.1, 44.1.
pr_ex_test: 43.1, 44.1.
pr_ex_test_check: 43.1.

pr_ex_test_dim: 43.1, 43.4.
pr_ex_test_lookup: 43.1.
pr_ex_test_num: 43.1, 43.4, 44.2.
pr_ex_ts_bk: 43.1, 44.1.
pr_ex_ts_rc: 43.1, 44.1.
pr_ex_ts_rc_num: 43.1, 44.1.
pr_exrc_dim: 43.4, 44.2.
pr_exrc_num: 43.4, 44.2.
pr_generic_args: 43.1.
pr_generic_copy: 43.1.
pr_generic_decl: 43.1.
pr_generic_dummy: 43.1.
pr_mat_ref: 43.1, 44.1.
pr_materials_args: 43.1.
pr_materials_check: 43.1.
pr_materials_copy: 43.1.
pr_materials_decl: 43.1.
pr_materials_dummy: 43.1.
pr_materials_num: 43.1, 43.4, 44.2.
pr_materials_sub: 44.2.
pr_max_equiv: 43.2, 43.5.
pr_max_lines: 43.2.
pr_num_arrangements: 43.1, 44.1.
pr_num_change_vars: 43.3.
pr_num_diag_vars: 43.3.
pr_pm_case_num: 43.1, 44.1.
pr_pm_cases: 43.1, 44.1.
pr_pm_num_arrange: 43.1, 44.1.
pr_pm_prod: 43.1, 44.1.
pr_pm_prod_mult: 43.1.
pr_pm_ref: 43.1, 44.1.
pr_pmi: 48.1, 48.2.
pr_pmi_args: 43.1.
pr_pmi_check: 43.1.
pr_pmi_copy: 43.1.
pr_pmi_decl: 43.1.
pr_pmi_dummy: 43.1.
pr_pmi_max: 43.2, 43.4.
pr_pmi_num: 8.1, 43.1, 43.4, 44.2.
pr_problem_sp_back: 43.3, 44.3.
pr_problem_sp_test: 43.3, 44.3.
pr_prod_mult: 43.1, 44.1.
pr_rc_num: 43.1, 44.1.
pr_reac: 46.1, 46.2.
pr_reaction: 43.1, 44.1.
pr_reaction_args: 43.1.
pr_reaction_check: 43.1.
pr_reaction_copy: 43.1.
pr_reaction_decl: 43.1.
pr_reaction_dim: 30.4, 43.4.
pr_reaction_dummy: 43.1.
pr_reaction_lookup: 43.1, 44.3.
pr_reaction_max: 43.2, 43.4.
pr_reaction_num: 8.1, 8.3, 43.1, 43.4, 44.2.
pr_tag_string: 43.4.
pr_tag_string_length: 43.2, 43.4.
pr_test: 8.1, 43.1, 44.1.
pr_test_args: 43.1.
pr_test_check: 8.1, 43.1.
pr_test_copy: 43.1.
pr_test_decl: 8.1, 43.1.
pr_test_dummy: 43.1.
pr_test_lookup: 8.3, 43.1, 44.3.
pr_test_num: 43.1, 43.4, 44.2, 44.3.
pr_ts_bk: 43.1, 44.1.
pr_ts_prod: 43.1, 44.1.
pr_ts_rc: 43.1, 44.1.
pr_var_angle: 43.3.
pr_var_back_index: 43.3.
pr_var_emitter_v_vector: 43.3.
pr_var_emitter_v_2: 43.3.
pr_var_emitter_v_3: 43.3.
pr_var_emitter_vf_Maxwell_vector: 43.3.
pr_var_emitter_vf_Maxwell_2: 43.3.
pr_var_emitter_vf_Maxwell_3: 43.3.
pr_var_emitter_vth_Maxwell: 43.3.
pr_var_energy: 43.3.
pr_var_energy_change: 43.3.
pr_var_energy_in: 43.3.
pr_var_energy_out: 43.3.
pr_var_mass: 43.3.
pr_var_mass_change: 43.3, 44.3.
pr_var_mass_in: 43.3.
pr_var_mass_out: 43.3.
pr_var_momentum_change_vector: 43.3.
pr_var_momentum_change_2: 43.3.
pr_var_momentum_change_3: 43.3.
pr_var_momentum_in_vector: 43.3.
pr_var_momentum_in_2: 43.3.
pr_var_momentum_in_3: 43.3.
pr_var_momentum_out_vector: 43.3.
pr_var_momentum_out_2: 43.3.
pr_var_momentum_out_3: 43.3.
pr_var_momentum_vector: 43.3.
pr_var_momentum_2: 43.3.
pr_var_momentum_3: 43.3.
pr_var_problem_sp_index: 30.3, 43.3, 44.3.
pr_var_unknown: 43.3.
pr_var_xy_stress: 43.3.
pr_var0_list: 31.1, 43.4, 44.2, 44.3, 46.1.
pr_var0_list_ind: 43.4.
pr_var0_num: 43.4, 44.2.
problem: 44.4.
problem_arr_ind: 43.4.

problem_background_ind: 43.4.
problem_background_reaction: 43.1, 43.4.
problem_background_sp: 43.1, 43.4.
problem_bkrc_ind: 43.4.
problem_bkrc_products: 43.1, 43.4.
problem_bkrc_reagents: 43.1, 43.4.
problem_bkrc_rg_ind: 43.4.
problem_ex_test_ind: 43.4.
problem_ex_test_sp: 43.1, 43.4.
problem_exrc_ind: 43.4.
problem_exrc_products: 43.1, 43.4.
problem_external_reaction: 43.1, 43.4.
problem_external_test_background: 43.1, 43.4.
problem_external_test_reaction: 43.1, 43.4.
problem_external_test_reaction_num: 43.1, 43.4.
problem_infile: 56.1.
problem_materials_ref: 43.1, 43.4.
problem_materials_ref_ind: 43.4.
problem_materials_sub: 43.4.
problem_materials_sub_ind: 43.4.
problem_num_arrangements: 43.1, 43.4.
problem_pmi_case_num: 43.1, 43.4.
problem_pmi_cases: 43.1, 43.4.
problem_pmi_ind0: 43.4.
problem_pmi_num_arrange: 43.1, 43.4.
problem_pmi_prod_mult: 43.1, 43.4.
problem_pmi_products: 43.1, 43.4.
problem_pmi_ref: 43.1, 43.4.
problem_pmi_ref_ind: 43.4.
problem_pmi_sub: 43.4, 44.2.
problem_pmi_sub_ind: 43.4, 47.2.
problem_prod_mult: 43.1, 43.4.
problem_product_ind: 43.4.
problem_rc: 43.1, 43.4.
problem_reaction_ind: 43.4, 45.2.
problem_reaction_ind0: 43.4.
problem_reaction_num: 43.1, 43.4.
problem_species_background: 43.1, 43.4.
problem_species_ind: 43.4, 43.5.
problem_species_test: 43.1, 43.4.
problem_test_background: 43.1, 43.4.
problem_test_ind: 43.4.
problem_test_products: 43.1, 43.4.
problem_test_reaction: 43.1, 43.4.
problem_test_sp: 43.1, 43.4.
problem_version: 43.4.
problemfile: 56.1.
problemsetup: 44.4.
process_pk: 50.7.
product_ind: 14.3.
ps: 43.5, 44.4.
psp: 43.3, 44.3.
pt: 9.1.
pt_args: 8.1, 8.3, 12.1.
pt_author: 8.1, 8.3, 9.1, 60.1.
pt_author_pm: 8.3, 9.2.
pt_author_rc: 8.3, 9.2.
pt_author_so: 8.3, 9.2.
pt_author1: 8.1.
pt_check: 8.1, 12.1.
pt_copy: 8.1, 12.1, 12.3, 60.1.
pt_decl: 8.1.
pt_decls: 8.1.
pt_dummy: 8.1, 12.1.
pt_geometry: 8.2, 8.3.
pt_init: 8.3, 9.2, 12.3.
pt_ion: 8.2, 8.3.
pt_loc: 8.1, 8.3, 9.1, 60.1.
pt_loc1: 8.1.
pt_neutral: 8.2, 8.3.
pt_scatter_thru: 8.3, 9.2.
pt_sp: 1, 8.1, 8.3, 9.1, 60.1.
pt_specular: 8.3, 9.2.
pt_sp1: 8.1.
pt_t: 8.1, 9.1, 60.1.
pt_test: 8.1, 8.3, 9.1, 60.1.
pt_test1: 8.1.
pt_thru_face: 8.3, 9.2.
pt_track: 8.3, 9.2.
pt_type: 8.1, 8.3, 9.1, 60.1.
pt_type1: 8.1.
pt_t1: 8.1.
pt_v: 8.1, 8.3, 9.1, 60.1.
pt_v1: 8.1.
pt_w: 8.1, 8.3, 9.1, 60.1.
pt_w1: 8.1.
ptr: 50.4, 51.
ptr_arrayname: 51.
ptr_decl: 50.2, 50.4.
ptr_foo: 51.
ptr_ind: 31.1.
ptr_junk_array: 50.2.
ptr_x: 50.4.
px: 60.1.
ra: 39.2, 40.1.
ra_bool_and_label: 39.2, 40.1.
ra_bool_not_label: 39.2, 40.1.
ra_data_filename: 39.2, 40.1.
ra_data_filename_length: 39.1, 39.2.
ra_dep_var_ind: 39.2.
ra_fit_coef_max: 39.1, 39.4.
ra_hier_label: 39.2, 40.1.
ra_label_max: 39.1, 39.2.
ra_localcommon: 39.4.

ra_mass: 39.2, 40.1.
ra_mass_units: 39.2, 40.1.
ra_num_bool_and: 39.2, 40.1.
ra_num_bool_not: 39.2, 40.1.
ra_num_hier: 39.2, 40.1.
ra_search_label_length: 39.1, 39.2.
ra_unit_string_length: 39.1, 39.2.
ragged_alloc: 30.6, 45.3.
ran_array_: 36.1.
ran_c: 22.3, 36.2, 54.3.
ran_index_: 36.1.
ran_k: 36.1, 36.2, 60.1, 60.2.
ran_s: 32.3, 33.1, 36.2, 36.3.
ran_temp: 36.1.
random: 36.1, 36.3.
random_cosdist: 36.3.
random_gauss: 36.3.
random_init_d2: 36.3.
random_isodist: 36.3.
range: 50.2.
range_max: 30.6, 37.5, 41.6, 45.3, 47.3, 50.2, 50.5.
range_max1: 50.2.
range_min: 30.6, 37.5, 41.6, 45.3, 47.3, 50.2, 50.3, 50.5.
range_min1: 50.2.
range_size: 50.2, 50.5, 52.2.
range_star: 50.2.
range_stara: 50.2.
rangea: 50.2.
rank: 48.1.
rank_: 52.2.
rank_ind: 31.1, 37.3, 38.1.
rank_ind0: 37.3, 38.1.
rank_x: 50.2.
ratecalc: 38.1.
rc: 9.2, 14.3, 15.1, 43.1, 44.1.
rc_args: 14.1.
rc_check: 14.1.
rc_copy: 14.1.
rc_decl: 14.1.
rc_dummy: 14.1.
rc_emitter: 14.1, 15.1.
rc_filename: 14.1, 15.1.
rc_gen: 14.1, 15.1.
rc_generic_no: 14.1, 15.1.
rc_generic_yes: 14.1, 15.1.
rc_lookup: 14.1, 15.2.
rc_name: 14.1, 15.1.
rc_name_len: 14.2, 14.3, 15.1.
rc_num: 14.1, 14.3, 38.1.
rc_product: 14.1, 15.1, 44.1.
rc_product_max: 14.2, 14.3, 15.1, 43.4.
rc_product_num: 14.1, 15.1.
rc_reaction_type: 14.1, 15.1.
rc_reagent: 1, 14.1, 15.1.
rc_reagent_max: 14.2, 14.3, 15.1.
rc_reagent_num: 14.1, 15.1.
rc_sy: 14.1, 15.1, 38.1.
rc_sy_len: 14.2, 14.3, 15.1, 37.3.
rc_type_len: 14.2, 14.3.
rd: 45.2, 46.1, 48.1.
rd_data: 46.2.
rd_data_args: 45.1, 46.2.
rd_data_base: 46.2.
rd_data_size: 46.2.
rd_data_tab: 46.2.
rd_decls: 45.3.
rd_dep_var_ind: 45.2.
rd_dep_var_max: 45.1, 45.2.
rd_eval_string: 45.2.
rd_handling: 45.1, 46.1, 46.2.
rd_handling_table_ind: 45.2.
rd_null_alloc: 45.3.
rd_ragged_alloc: 45.3, 46.1, 46.2.
rd_ragged_realloc: 45.3, 46.1, 46.2.
rd_rank_ind: 45.2.
rd_rank_ind0: 45.2.
rd_rate: 45.1, 46.1, 46.2.
rd_set_baseinc: 45.3.
rd_tab_index: 46.2.
rd_table_ind: 45.2.
rd_tag_string: 45.2.
reac: 8.3.
reac_so: 31.1.
react_begin: 39.4.
react_done: 39.4.
reaction: 15.1.
reaction_emitter: 14.1, 14.3.
reaction_filename: 14.1, 14.3.
reaction_filename_string: 14.3.
reaction_generic: 14.1, 14.3.
reaction_handling: 1, 46.1, 46.2.
reaction_handling_base: 45.1, 45.2, 46.1.
reaction_handling_delta: 45.2, 46.1.
reaction_handling_eval_name: 45.2, 46.1.
reaction_handling_min: 45.2, 46.1.
reaction_handling_num_rand: 45.2, 46.1.
reaction_handling_rank: 45.2, 46.1.
reaction_handling_size: 45.2, 46.1.
reaction_handling_spacing: 45.2, 46.1.
reaction_handling_tab: 45.1, 45.2, 46.1.
reaction_handling_tab_index: 45.1, 45.2, 46.1.
reaction_handling_var: 45.2, 46.1.
reaction_handling_var0: 31.1, 45.2, 46.1.

reaction_ind: 1, 14.3.
reaction_infile: 56.1.
reaction_name: 14.1, 14.3.
reaction_name_string: 14.3.
reaction_product: 14.1, 14.3.
reaction_product_num: 14.1, 14.3.
reaction_rate: 46.1, 46.2.
reaction_rate_base: 45.1, 45.2, 46.1.
reaction_rate_delta: 45.2, 46.1.
reaction_rate_eval_name: 45.2, 46.1.
reaction_rate_min: 45.2, 46.1.
reaction_rate_num_rand: 45.2, 46.1.
reaction_rate_rank: 45.2, 46.1.
reaction_rate_size: 45.2, 46.1.
reaction_rate_spacing: 45.2, 46.1.
reaction_rate_tab: 45.1, 45.2, 46.1.
reaction_rate_tab_index: 45.1, 45.2, 46.1.
reaction_rate_var: 45.2, 46.1.
reaction_reagent: 1, 14.1, 14.3.
reaction_reagent_num: 14.1, 14.3.
reaction_sy: 14.1, 14.3.
reaction_symbol_string: 14.3.
reaction_type: 14.1, 14.3.
reaction_type_string: 14.3.
reaction_version: 14.3.
reactionfile: 56.1.
read_int_soft_fail: 49.
read_integer: 49.
read_real: 49.
read_real_scaled: 49.
read_real_soft_fail: 49.
read_string: 49.
read_text: 49.2.
readbackgroundd: 23.3.
readfilenames: 57.
reagent.ind: 1, 14.3.
real_alloc: 50.4.
real_free: 50.4.
real_realloc: 50.4.
recyc: 26.3, 27.2.
register: 50.6.
rel_wt: 60.1.
rf: 56.2.
rn_args: 12.1, 36.1, 36.3.
rn_array: 36.1, 60.1.
rn_array1: 36.1.
rn_copy: 12.1, 36.1.
rn_cos_next: 36.3.
rn_decl: 12.1, 36.1.
rn_decls: 36.1.
rn_dummy: 12.1, 36.1.
rn_gauss_next: 36.3.
rn_index: 36.1, 60.1.
rn_index1: 36.1.
rn_init: 36.3.
rn_iso_next: 36.3.
rn_next: 36.3.
rn_seed_args: 36.3.
rn_seed_copy: 36.3.
rn_seed_copy1: 36.3.
rn_seed_decl: 36.1, 36.3.
rsp: 12.3, 13.2, 60.1.
rx: 60.1.
sa: 34.2, 35.1.
sample_sources: 23.2.
sc: 26.2, 26.3, 27.1.
sc_args: 26.1.
sc_check: 26.1, 27.2.
sc_compare: 26.1, 27.2.
sc_copy: 26.1.
sc_decl: 26.1.
sc_diag_angle: 26.1, 27.1.
sc_diag_check: 26.1, 27.2.
sc_diag_energy: 26.1, 27.1.
sc_diag_ind: 26.2.
sc_diag_max_bins: 26.2, 27.1, 29.1.
sc_diag_name_len: 26.1, 26.2.
sc_diag_name_string: 26.2.
sc_diag_size: 26.2, 27.1.
sc_diag_spacing_linear: 26.1.
sc_diag_spacing_log: 26.1.
sc_diag_spacing_unknown: 26.1.
sc_diag_unknown: 26.1.
sc_diagnostic: 26.1, 27.2.
sc_diagnostic_grps: 26.1, 26.2, 27.1.
sc_dummy: 26.1.
sc_exit: 26.1, 26.3.
sc_exit_check: 26.1, 27.2.
sc_exit_num: 26.1, 26.2, 26.3, 27.1.
sc_neg: 26.2, 26.3.
sc_plasma: 26.1, 26.3, 27.2.
sc_plasma_check: 26.1, 27.2.
sc_plasma_decl: 26.1.
sc_plasma_num: 26.1, 26.2, 26.3, 27.1.
sc_pos: 26.2, 26.3.
sc_target: 26.1, 26.3.
sc_target_check: 26.1, 27.2.
sc_target_decl: 26.1.
sc_target_num: 26.1, 26.2, 26.3, 27.1.
sc_type_max: 26.1, 26.2.
sc_unknown: 26.1.
sc_vacuum: 26.1, 26.3.
sc_vacuum_check: 26.1, 27.2.
sc_vacuum_num: 26.1, 26.2, 26.3, 27.1.

sc_wall: [26.1](#), 26.3, 27.2.
sc_wall_check: [26.1](#), 27.2.
sc_wall_decl: [26.1](#).
sc_wall_num: [26.1](#), 26.2, 26.3, 27.1.
scal.ind: 31.1.
scoring_data: 30.3, 31.2.
scoring_data_max: [30.3](#).
se: 29.1.
search_label: 39.2.
sect: 26.1, 27.2.
sector: 26.1, 27.1, 27.2.
sector_: 10.
sector_ind: 26.2.
sector_neg_pos_ind: 26.2.
sector_next_: 10.
sector_points: 26.2, 27.1.
sector_strata_segment: 26.1, 26.2, 27.1.
sector_surface: 26.1, 26.2, 27.1.
sector_type_ind: 26.2.
sector_type_pointer: 26.2, 26.3, 27.1.
sector_zone: 26.1, 26.2, 27.1.
sectors: 26.2, 27.1.
seed: 36.3.
seed_ind: 33.1.
seg: 26.1, 27.2.
set_base: [37.5](#), [41.6](#).
set_inc: [37.5](#), [41.6](#).
set_prob_alias: 23.2.
set_var_list: 30.3, 31.1.
single_precision: 36.1.
size: 50.2.
slice1: [50.3](#).
sn: 54.3, 55.1.
sn_float: 60.1.
sn_float_lc_x1: [54.2](#).
sn_float_lc_x2: [54.2](#).
sn_float_lc_x3: [54.2](#).
sn_float_pt_t: [54.2](#).
sn_float_pt_v1: [54.2](#).
sn_float_pt_v2: [54.2](#).
sn_float_pt_v3: [54.2](#).
sn_float_pt_w: [54.2](#).
sn_int_lc_cell: [54.2](#).
sn_int_lc_cell_next: [54.2](#).
sn_int_lc_face: [54.2](#).
sn_int_lc_sector: [54.2](#).
sn_int_lc_sector_next: [54.2](#).
sn_int_lc_zone: [54.2](#).
sn_int_lc_zone_next: [54.2](#).
sn_int_pt_author: [54.2](#).
sn_int_pt_sp: [54.2](#).
sn_int_pt_test: [54.2](#).
sn_int_pt_type: [54.2](#).
sn_number_particles: 54.3, 55.1.
sn_particles_dim: 54.3.
sn_particles_float: 54.3, 55.1.
sn_particles_int: 54.3, 55.1.
sn_pt_float_ind: 54.3.
sn_pt_float_max: [54.2](#), 54.3.
sn_pt_int_ind: 54.3.
sn_pt_int_max: [54.2](#), 54.3.
sn_seed_decimal: 54.3, 55.1.
sn_seed_decimal_ind: 54.3.
snapshot_pdf_ind: 54.3.
snapshotfile: 23.1, [56.1](#).
so: 22.3, 23.1, 44.1.
so_args: [22.1](#).
so_base: [22.1](#), 23.1.
so_check: [22.1](#).
so_chkpt: [22.1](#), 23.1.
so_copy: [22.1](#).
so_decl: [22.1](#).
so_delta_fn: [22.1](#), 23.1.
so_direct: [22.1](#), 23.3.
so_direct_delta: 22.3.
so_direct_mult: [22.1](#).
so_dummy: [22.1](#).
so_e_bins_num_max: [22.1](#).
so_e_bins_spacing_linear: [22.1](#).
so_e_bins_spacing_log: [22.1](#).
so_e_bins_spacing_unknown: [22.1](#).
so_geom: [22.1](#), 23.1.
so_giparam_e_bins_num: [22.1](#).
so_giparam_e_bins_spacing: [22.1](#).
so_giparams_data: [22.1](#), 23.4.
so_giparams_list: [22.1](#), 23.4.
so_giparams_list_dim: 22.3, 23.4.
so_giparams_list_size: 22.3, 23.4.
so_gparam_e_bins_delta: [22.1](#).
so_gparam_e_bins_min: [22.1](#).
so_gparam_puff_exponent: [22.1](#).
so_gparam_puff_temp: [22.1](#), 23.4.
so_gparam_unknown: [22.1](#).
so_gparams_data: [22.1](#), 23.4.
so_gparams_list: [22.1](#), 23.4.
so_gparams_list_dim: 22.3, 23.4.
so_gparams_list_size: 22.3, 23.4.
so_grps: 22.1, 22.3, 23.2, 32.3.
so_iparam_lc_cell: [22.1](#).
so_iparam_lc_zone: [22.1](#).
so_iparam_pt_author: [22.1](#).
so_iparam_pt_sp: [22.1](#).
so_iparam_pt_test: [22.1](#).
so_iparam_pt_type: [22.1](#).

so_iparams_data: 22.1, 23.4.
so_iparams_data_dim: 22.3, 23.4.
so_iparams_data_size: 22.3, 23.4.
so_iparams_list: 22.1, 23.4.
so_iparams_list_dim: 22.3, 23.4.
so_iparams_list_size: 22.3, 23.4.
so_line: 22.1.
so_name: 22.1, 22.2, 23.2.
so_name_ind: 22.3.
so_name_len: 22.1, 22.3.
so_nflights: 22.1, 23.1.
so_nseg: 22.1, 23.1.
so_param_e_bin_prob: 22.1.
so_param_e_ion_delta: 22.1.
so_param_e_ion_mult: 22.1.
so_param_e_ion_sheath: 22.1.
so_param_lc_x1: 22.1.
so_param_lc_x2: 22.1.
so_param_lc_x3: 22.1.
so_param_pt_t: 22.1.
so_param_pt_v1: 22.1.
so_param_pt_v2: 22.1.
so_param_pt_v3: 22.1.
so_param_temperature: 22.1, 23.4.
so_param_unknown: 22.1.
so_param_v1: 22.1.
so_param_v2: 22.1.
so_param_v3: 22.1.
so_params_data: 22.1, 23.4.
so_params_data_dim: 22.3, 23.4.
so_params_data_size: 22.3, 23.4.
so_params_list: 22.1, 23.4.
so_params_list_dim: 22.3, 23.4.
so_params_list_size: 22.3, 23.4.
so_plate: 22.1, 22.2.
so_plt_e_bins: 22.1, 22.2.
so_point: 22.1.
so_puff: 22.1, 22.2.
so_random: 22.1, 22.2, 23.3.
so_recomb: 22.1, 22.2.
so_rel_wt_max: 22.2, 22.3, 23.3.
so_rel_wt_min: 22.2, 22.3, 23.3.
so_restart: 22.2, 22.3, 23.3.
so_root_sp: 13.1, 22.1, 23.1.
so_sampling: 22.2, 22.3, 23.3.
so_scale: 22.1, 23.1.
so_seed_decimal: 22.2, 22.3, 23.3.
so_seed_decimal_ind: 22.3.
so_seed_spacing: 22.2, 22.3, 23.3.
so_seg_check: 22.1.
so_seg_tot: 22.1, 22.3, 23.2, 23.4.
so_set_run_flags: 22.2, 23.3.
so_snapshot: 22.1, 22.2.
so_spaced_seeds: 22.2, 22.3, 23.3, 33.1.
so_species: 22.1, 23.1.
so_surface: 22.1.
so_t_varn: 22.1.
so_time_dependent: 22.2, 22.3, 23.3.
so_time_final: 22.2, 22.3, 23.1, 23.3, 31.1.
so_time_initial: 22.2, 22.3, 23.1, 23.3.
so_time_INITIALIZATION: 22.2, 22.3, 23.3.
so_time_uniform: 22.1, 23.1.
so_time_varn: 23.1.
so_tot_curr: 22.1, 23.1.
so_type: 8.3, 13.1, 22.1, 23.1.
so_type_ind: 22.3.
so_type_num: 8.1, 8.3, 22.1, 22.3, 30.4.
so_vol_source: 22.1, 22.2.
so_volume: 22.1.
so_wt_norm: 22.1, 23.1, 23.2, 23.3.
so_wt_norm_max: 22.2, 22.3, 23.3.
so_wt_norm_min: 22.2, 22.3, 23.3.
source: 8.3, 9.2, 50.7.
source_: 12.1.
source_base_ptr: 22.1, 22.3.
source_current: 22.3, 23.2.
source_geometry: 22.1, 22.3.
source_giparameters_base: 22.1, 22.3, 23.4.
source_giparameters_data: 22.1, 22.3, 23.4.
source_giparameters_list: 22.1, 22.3, 23.4.
source_giparams_ind: 22.3.
source_gparameters_base: 22.1, 22.3.
source_gparameters_data: 22.1, 22.3.
source_gparameters_list: 22.1, 22.3.
source_gparams_ind: 22.3.
source_grp_ind: 22.3.
source_iparameters_base: 22.1, 22.3, 23.4.
source_iparameters_data: 22.1, 22.3, 23.4.
source_iparameters_data_base: 22.1, 22.3, 23.4.
source_iparameters_list: 22.1, 22.3, 23.4.
source_iparams_data_ind: 22.3.
source_iparams_ind: 22.3.
source_kseg_: 12.1.
source_name: 22.1, 22.3.
source_num_checkpoints: 22.1, 22.3.
source_num_flights: 22.1, 22.3.
source_num_giparameters: 22.3, 23.4.
source_num_gparameters: 22.3, 23.4.
source_num_iparameters: 22.1, 22.3, 23.4.
source_num_parameters: 22.1, 22.3, 23.4.
source_num_segments: 22.1, 22.3.
source_parameters_base: 22.1, 22.3.
source_parameters_data: 22.1, 22.3.
source_parameters_data_base: 22.1, 22.3.

source_parameters_list: 22.1, 22.3.
source_params_data_ind: 22.3.
source_params_ind: 22.3.
source_root_sp_: 12.1.
source_root_species: 22.1, 22.3.
source_scale_factor: 22.1, 22.3.
source_seg_ind: 22.3.
source_segment_prob_alias: 22.3, 23.2.
source_segment_ptr: 13.1, 22.3, 23.2.
source_segment_ptr_alias: 22.3, 23.2.
source_segment_rel_wt: 22.3, 23.1, 23.2, 23.3.
source_species: 22.1, 22.3.
source_time_variation: 22.1, 22.3.
source_total_current: 22.1, 22.3.
source_type: 22.1, 22.3.
source_type_: 12.1.
source_weight_norm: 22.1, 22.3.
source_xseg_: 12.1.
sp: 6.3, 7.1, 19.1, 43.1, 44.1.
SP: [49.1](#).
sp_args: [6.1](#).
sp_check: [6.1](#), 8.1.
sp_copy: [6.1](#).
sp_count: [6.1](#), 7.1.
sp_decl: [6.1](#).
sp_dummy: [6.1](#).
sp_el: 1, [6.1](#), 7.1.
sp_generic: [6.1](#), 7.1.
sp_lookup: [6.1](#), 7.2, 8.3.
sp_m: [6.1](#), 7.1, 8.3.
sp_multiplicity: [6.1](#), 7.1.
sp_name: [6.1](#), 7.1.
sp_name_len: [6.2](#), 6.3, 7.1.
sp_ncomp: [6.1](#), 7.1.
sp_ncomp_max: [6.2](#), 6.3, 7.1.
sp_num: 6.1, 6.3, 43.4.
sp_sy: [6.1](#), 7.1.
sp_sy_len: [6.2](#), 6.3, 7.1.
sp_z: [6.1](#), 7.1, 8.3.
species: 7.1.
species_: 8.1.
species_comp_ind: 6.3.
species_count: 6.1, 6.3.
species_el: 6.1, 6.3.
species_generic: 6.1, 6.3.
species_ind: 6.3.
species_infile: [56.1](#).
species_m: 6.1, 6.3.
species_multiplicity: 6.1, 6.3.
species_name: 6.1, 6.3.
species_name_string: 6.3.
species_ncomp: 6.1, 6.3.
species_sy: 6.1, 6.3.
species_symbol_string: 6.3.
species_version: 6.3.
species_z: 6.1, 6.3.
speciesfile: [56.1](#).
sqrt: 20.2, 20.3, 22.1.
src: 57.
st_decls: [49](#).
stack_: 12.1.
start_: 50.7.
start_common: [50.7](#).
start_module: [50.7](#).
start_mpibcast: [50.7](#).
start_mpibcastna: [50.7](#).
start_mpireceive: [50.7](#).
start_mpisend: [50.7](#).
start_ncdecl: [50.7](#).
start_ncdef: [50.7](#).
start_ncread: [50.7](#).
start_ncwrite: [50.7](#).
start_pvmget: [50.7](#).
start_pvmput: [50.7](#).
stat_comp_ff: 35.1.
stat_comp_fin: 34.2.
stat_comp_flt: 34.2.
stat_comp_frag: 34.2.
stat_dim_ff: 35.1.
stat_dim_fin: 34.2.
stat_dim_flt: 34.2.
stat_dim_frag: 34.2.
stat_ff: 35.1.
stat_fin: 34.2, 35.1.
stat_fin_ind: 34.2.
stat_flt: 34.2.
stat_flt_ind: 34.2.
stat_frag: 34.2.
stat_frag_ind: 34.2.
stat_mem_inc_ff: 35.2.
stat_mem_inc_fin: [34.1](#).
stat_mem_inc_flt: [34.1](#).
stat_mem_inc_frag: [34.1](#).
stat_moments_ind: 34.2.
stat_pf_dim_ff: 35.1.
stat_pf_dim_fin: 34.2.
stat_pf_dim_flt: 34.2.
stat_pf_dim_frag: 34.2.
stat_pf_fin_ind: 34.2.
stat_pf_flt_ind: 34.2.
stat_pf_frag_ind: 34.2.
stat_ps_dim_ff: 35.1.
stat_ps_dim_fin: 34.2.
stat_ps_dim_flt: 34.2.

stat_ps_dim_frag: 34.2.
stat_ps_fin_ind: 34.2.
stat_ps_flt_ind: 34.2.
stat_ps_frag_ind: 34.2.
stat_ptr2full_fff: 35.1.
stat_ptr2full_fin: 34.2.
stat_ptr2full_flt: 34.2.
stat_ptr2full_frag: 34.2.
stat_ptr2short_fff: 35.1.
stat_ptr2short_fin: 34.2.
stat_ptr2short_flt: 34.2.
stat_ptr2short_frag: 34.2.
stat_size_fff: 35.1.
stat_size_fin: 34.2.
stat_size_flt: 34.2.
stat_size_frag: 34.2.
stat_wt_dim_fff: 35.1.
stat_wt_dim_fin: 34.2.
stat_wt_dim_frag: 34.2.
stat_wt_fff: 35.1.
stat_wt_fin: 34.2.
stat_wt_fin_ind: 34.2.
stat_wt_frag: 34.2, 35.1.
stat_wt_frag_ind: 34.2.
stat_wt_tot_fff: 35.1.
stat_wt_tot_fin: 34.2.
stat_wt_tot_flt: 34.2.
stat_wt_tot_frag: 34.2.
strata: 26.1, 26.2, 27.1.
stratum: 26.1, 27.2.
string: 4.3, 6.3, 14.3, 16.3, 18.2, 24.3, 30.4, 32.3, 37.3, 41.4, 43.4.
string_length: 49, 49.1.
string_lookup: 4.1, 6.1, 14.1, 16.1, 18, 26.1, 28.1, 49.
subs: 30.6, 45.1, 45.3, 46.2, 47.1, 47.3, 48.2.
surface: 27.1.
surface_: 10.
surface_reflect: 8.3.
surface_sectors: 27.1.
surface_specular: 8.3.
sy: 4.1, 6.1, 8.3, 9.2, 12.3, 13.2, 14.1, 16.1, 18.
sym: 20.2, 20.3.
symbol: 5.2, 7.2, 15.2, 17.3, 19.3.
t_sector: 27.2.
tab_index: 48.1.
table_calc: 38.1.
table_eval: 38.1.
table_external: 38.1.
table_ind: 33.1.
tag: 50.7.
tag_string: 37.3.
tally: 31.1.
tally_base: 30.4, 31.1, 32.2.
tally_cv_action: 30.4, 31.1.
tally_cv_ind: 30.4.
tally_cv_num_partners: 30.4, 31.1.
tally_cv_partner: 31.1.
tally_cv_partner_ind: 30.4.
tally_cv_partners: 30.4.
tally_cv_ptr: 30.4, 31.1.
tally_cv_ptr_ind: 30.4.
tally_cv_scaler: 30.4.
tally_cv_scalers: 30.4, 31.1.
tally_cv_type: 30.4, 31.1.
tally_dep_var: 30.4, 31.1.
tally_dep_var_dim: 30.3, 30.4, 31.1, 31.2, 32.2, 33.2.
tally_est_ind: 30.4.
tally_est_reaction: 30.4, 31.1.
tally_est_test: 30.4, 31.1.
tally_geometry: 30.4, 31.1.
tally_geometry_ptr: 30.4, 31.1.
tally_ind: 30.4, 31.1.
tally_indep_var: 30.4, 31.1.
tally_index_ind: 30.4.
tally_infile: 56.1.
tally_name: 30.4, 31.1.
tally_name_string: 30.4.
tally_num_conversions: 30.4, 31.1.
tally_rank: 30.4, 31.1, 33.2.
tally_rank_ind: 30.4.
tally_reac_ind: 30.4.
tally_size: 30.4, 31.1, 32.3, 35.1.
tally_tab_index: 30.4, 31.1, 32.2, 33.2.
tally_tag_string: 30.4.
tally_type: 30.4, 31.1.
tally_type_base: 30.4, 31.1.
tally_type_ind: 30.4.
tally_type_num: 30.4, 31.1.
tally_var_list: 30.4.
tally_version: 30.4.
tallyfile: 56.1.
tallysetup: 31.1.
target: 26.2, 27.1.
target_ind: 26.2, 27.1.
target_lookup: 26.1, 27.2.
target_material: 26.2, 26.3, 27.1.
target_recyc_coeff: 26.2, 26.3, 27.1.
target_sector: 26.1, 26.2, 26.3, 27.1.
target_temperature: 26.2, 26.3, 27.1.
temp: 26.3, 27.2.
TEMPCOUNT: 50.7.
temperature: 39.4.
test: 43.3, 44.3.
test_: 8.1.

testfile: [56.1.](#)
time_: [8.1.](#)
tl: [30.4, 31.1.](#)
tl_args: [30.1.](#)
tl_check: [30.1.](#)
tl_copy: [30.1.](#)
tl_cv_divide_number: [30.3, 31.1.](#)
tl_cv_max_conversions: [30.3, 30.4, 31.1.](#)
tl_cv_max_local_data: [30.3.](#)
tl_cv_max_params: [30.3.](#)
tl_cv_max_partners: [30.3, 30.4, 31.1.](#)
tl_cv_max_scalers: [30.3, 30.4, 31.1.](#)
tl_cv_output: [30.3, 31.1.](#)
tl_cv_Pa_per_mTorr: [30.3.](#)
tl_cv_partner_unknown: [30.3.](#)
tl_cv_pos_1: [30.3.](#)
tl_cv_pos_2: [30.3.](#)
tl_cv_pos_3: [30.3.](#)
tl_cv_post: [30.3, 31.1.](#)
tl_cv_problem_sp_mass: [30.3.](#)
tl_cv_scale: [30.3, 31.1.](#)
tl_cv_scaler_unknown: [30.3.](#)
tl_cv_test_mass: [30.3.](#)
tl_cv_three_halves: [30.3.](#)
tl_cv_to_external_coords: [30.3, 31.1.](#)
tl_cv_to_internal_coords: [30.3, 31.1.](#)
tl_cv_track: [30.3, 31.1.](#)
tl_cv_unknown: [30.3.](#)
tl_cv_volume: [30.3.](#)
tl_cv_zone_pos_1: [30.3.](#)
tl_cv_zone_pos_2: [30.3.](#)
tl_cv_zone_pos_3: [30.3.](#)
tl_decl: [30.1.](#)
tl_decls: [30.5.](#)
tl_dep_var_scalar: [30.3, 31.1.](#)
tl_dep_var_vector: [30.3, 31.1.](#)
tl_dummy: [30.1.](#)
tl_est_collision: [30.2, 31.1.](#)
tl_est_max: [30.2, 30.4.](#)
tl_est_post_process: [30.2, 31.1.](#)
tl_est_snapshot: [30.2, 31.1.](#)
tl_est_track: [30.2, 31.1.](#)
tl_est_unknown: [30.2, 31.1.](#)
tl_geom_detector: [30.3, 31.1.](#)
tl_geom_global: [30.3, 31.1.](#)
tl_geom_surface: [30.3, 31.1.](#)
tl_geom_unknown: [30.3.](#)
tl_geom_volume: [30.3, 31.1.](#)
tl_index: [31.1.](#)
tl_index_angle_bin: [30.3.](#)
tl_index_detector: [30.3.](#)
tl_index_diagnostic: [30.3.](#)
tl_index_energy_bin: [30.3.](#)
tl_index_material: [30.3.](#)
tl_index_max: [30.3, 30.4.](#)
tl_index_plasma_zone: [30.3.](#)
tl_index_pmi: [30.3.](#)
tl_index_problem_sp: [30.3.](#)
tl_index_reaction: [30.3.](#)
tl_index_sector: [30.3.](#)
tl_index_source_group: [30.3.](#)
tl_index_strata: [30.3.](#)
tl_index_strata_segment: [30.3.](#)
tl_index_test: [30.3.](#)
tl_index_test_author: [30.3.](#)
tl_index_unknown: [30.3.](#)
tl_index_wavelength_bin: [30.3.](#)
tl_index_zone: [30.3.](#)
tl_index_zone_ind_1: [30.3.](#)
tl_index_zone_ind_2: [30.3.](#)
tl_name_length: [30.3, 30.4.](#)
tl_num: [30.4, 31.1.](#)
tl_rank_max: [30.2, 30.4, 33.2.](#)
tl_scoring_index: [30.3, 31.2.](#)
tl_set_base_size: [30.6.](#)
tl_set_baseinc: [30.6.](#)
tl_tag_length: [30.3, 30.4.](#)
tl_type_est: [31.1.](#)
tl_type_max: [30.2, 30.4.](#)
tl_type_reaction: [30.2, 31.1.](#)
tl_type_sector: [30.2, 31.1.](#)
tl_type_test: [30.2, 31.1.](#)
tl_type undefined: [30.2.](#)
tmax: [8.3, 9.2.](#)
trim: [49.1.](#)
TRUE: [23.3, 33.1, 35.1.](#)
type: [8.3, 9.2, 12.3, 13.2, 24.4, 27.1, 50.2, 50.6, 51, 60.1.](#)
type_: [8.1.](#)
type_ind: [31.1.](#)
type_x: [50.2.](#)
u: [39.4.](#)
unit_string: [37.3.](#)
up: [39.4.](#)
use: [50.7.](#)
v_ext_to_int: [20.2.](#)
v_int_to_ext: [20.3.](#)
v_sector: [27.2.](#)
v_t: [20.2, 20.3.](#)
vacuum: [26.2, 27.1.](#)
vacuum_ind: [26.2, 27.1.](#)
vacuum_sector: [26.2, 26.3, 27.1.](#)
var: [48.1, 50.7.](#)

var_alloc: 37.5, 41.6, 45.3, 47.3, [50.4](#), 51.
var_beg: [50.2](#).
var_def: [52.2](#), 53.
var_free: [50.4](#), 51.
var_inq: [52.2](#), 53.
var_read: [52.2](#), 53.
var_realloc: 37.5, 41.6, 45.3, 47.3, [50.4](#), 50.5, 51.
var_realloca: 26.3, [50.5](#), 51.
var_reallocb: 50.5, 51.
var_reallocc: [50.5](#), 51.
var_type: [50.2](#).
var_type_star: [50.2](#).
var_typea: 50.2, 50.4.
var_write: [52.2](#), 53.
varname: 51, 53.
vc_copy: 8.1, 10, 20.2, 20.3.
vc_set: 60.1.
vector: 20.4, 24.3, 26.2, 28.3.
velocity: 8.1.
view_ind: 29.1.
vol_source: 23.1.

w_sector: 27.2.
wall: 26.2, 27.1.
wall_ind: 26.2, 27.1.
wall_lookup: [26.1](#), 27.2.
wall_material: 26.2, 26.3, 27.1.
wall_recyc_coef: 26.2, 26.3, 27.1.
wall_sector: 26.1, 26.2, 26.3, 27.1.
wall_temperature: 26.2, 26.3, 27.1.
walls: 26.2.
web: 23.3, 27.1, 31.1, 38.1, 42.1, 44.4, 48.1, 57.
weight: 8.1.

x: [4.1](#), [6.1](#), [14.1](#), [16.1](#), [18](#), [20.1](#), [22.1](#), [24.1](#), [26.1](#),
[28.1](#), [30.1](#), [36.3](#), [43.1](#).
x_base: 41.6, 47.3.
x_inc: 41.6.
x_size: 41.6, 47.3.
x_tab: 41.6, 47.3.
xs: 15.1, 37.3, 38.1, 46.1, 60.1.
xs_data: 38.2.
xs_data_base: 37.2, 37.3, 38.1.
xs_data_inc: 37.3, 38.1.
xs_data_ind: 37.3.
xs_data_size: 37.3, 38.1.
xs_data_tab: 37.2, 37.3, 38.1, 38.2.
xs_data_table: [37.2](#), 38.2.
xs_decls: [37.4](#).
xs_dep_var_max: [37.1](#), 37.3, 39.2, 40.1, 45.1.
xs_eval_name: 37.3, 38.1.
xs_eval_name_length: [37.1](#), 37.3, 39.4, 45.2.
xs_max: 37.3, 38.1.

xs_max_indep_params: [37.1](#).
xs_max_random: [37.1](#).
xs_min: 37.3, 38.1.
xs_mult: 37.3, 38.1.
xs_name: 37.3, 38.1.
xs_num_dep_var: 37.3, 38.1.
xs_ragged_alloc: [37.5](#), 38.2.
xs_ragged_realloc: [37.5](#), 38.2.
xs_rank: 37.3, 38.1.
xs_spacing: 37.3, 38.1.
xs_spacing_linear: [37.1](#), 46.1.
xs_spacing_log: [37.1](#), 46.1.
xs_spacing_unknown: [37.1](#).
xs_symbol_string: 37.3.
xs_tab_index: 37.2, 37.3, 38.1, 38.2, 46.2.
xs_table_rank_max: [37.1](#), 37.3, 45.2.
xs_tag_string_length: [37.1](#), 37.3, 45.2.
xs_unit_string_length: [37.1](#), 37.3.
xs_units: 37.3, 38.1.
xs_var: 37.3, 38.1.
xs_var_density: [37.1](#).
xs_var_elec_temperature: [37.1](#).
xs_var_energy: [37.1](#).
xs_var_m_back: [37.1](#).
xs_var_m_emitter: [37.1](#).
xs_var_m_test: [37.1](#).
xs_var_sp_energy: [37.1](#).
xs_var_sp_temperature: [37.1](#).
xs_var_temperature: [37.1](#).
xs_var_unknown: [37.1](#).
xs_var_v_flow_1: [37.1](#).
xs_var_v_flow_2: [37.1](#).
xs_var_v_flow_3: [37.1](#).
xs_var_v_flowb_1: [37.1](#).
xs_var_v_flowb_2: [37.1](#).
xs_var_v_flowb_3: [37.1](#).
xs_var_v_test_1: [37.1](#).
xs_var_v_test_2: [37.1](#).
xs_var_v_test_3: [37.1](#).
xs_var_zone: [37.1](#).
xs_var_1st_random_number: [37.1](#).
xsection: 46.1.
xsection_version: 37.3.
xseg: 12.3, 13.2.
xxx_base: 46.1, 48.1.
xz_ind: 59.1.

zero: 20.2, 20.3.
zi_ix: [24.2](#).
zi_iy: [24.2](#).
zi_iz: [24.2](#).
zi_ptr: [24.2](#), 25.1.
zn: 24.3, 25.1, 27.1.

zn_args: [24.1](#).
zn_check: [24.1](#).
zn_copy: [24.1](#).
zn_decl: [24.1](#).
zn_decls: [24.1](#).
zn_dummy: [24.1](#).
zn_exit: [24.2](#).
zn_index: [24.1](#), 25.1, 25.2, 27.1, 33.1.
zn_index_max: [24.2](#), 24.3, 25.1, 25.2.
zn_num: 24.3, 25.2.
zn_plasma: 20.4, [24.2](#), 25.1, 60.1.
zn_pointer: 20.1, [24.1](#), 24.4, 25.1.
zn_solid: [24.2](#), 25.1.
zn_type: [24.1](#), 24.4, 25.1, 60.1.
zn_type_max: [24.2](#), 24.3.
zn_type_set: [24.4](#).
zn_undefined: [24.2](#).
zn_vacuum: [24.2](#), 25.1, 60.1.
zn_volume: [24.1](#), 25.1.
zone: 24.4, 25.1, 25.2, 26.1, 27.1, 27.2.
zone_: 10.
zone_center: 24.3, 25.2.
zone_exit: [8.2](#).
zone_frags_ind: 29.1.
zone_ind: 24.3.
zone_index: 24.1, 24.2, 24.3, 26.2, 33.1.
zone_index_ind: 24.3.
zone_index_max: 24.3, 25.2.
zone_index_min: 24.3, 25.2.
zone_max: 24.3, 25.2.
zone_min: 24.3, 25.2.
zone_next_: 10.
zone_pointer: 24.1, 24.3.
zone_type: 24.1, 24.3.
zone_type_ind: 24.3.
zone_type_num: 20.4, 24.3, 24.4.
zone_version: 24.3.
zone_volume: 24.1, 24.3.

COMMAND LINE: "fweave -f -i! -W[-ybs15000 -ykw800 -ytw40000 -j -n/
/Users/dstotler/degas2/src/classes.web".

WEB FILE: "/Users/dstotler/degas2/src/classes.web".

CHANGE FILE: (none).

GLOBAL LANGUAGE: FORTRAN.